App Inventor

Seus primeiros aplicativos Android

Casa do Código

NELSON FABBRI GERBELLI
VALÉRIA HELENA P. GERBELLI

ISBN

Impresso e PDF: 978-85-94188-30-4

EPUB: 978-85-94188-31-1

MOBI: 978-85-94188-32-8

Caso você deseje submeter alguma errata ou sugestão, acesse http://erratas.casadocodigo.com.br.

SOBRE OS AUTORES

Nelson Fabbri Gerbelli

Nelson é formado em nível técnico em Processamento de Dados em 1988, e concluiu a graduação na área de Tecnologia em Processamento de Dados em 1992. Concluiu também Análise de Sistemas em nível de especialização em 1994, licenciatura plena em Informática no ano de 1998 e licenciatura plena em Matemática no ano de 2007. Em 2017, terminou seu curso de Pedagogia.

É professor em cursos de TI desde 1995, atuando desde o ano de 2001 no Centro Estadual de Educação Tecnológica Paula Souza, nas ETECs do Estado de São Paulo onde já foi coordenador de cursos técnicos por vários anos. Em 2007, ingressou na Prefeitura do Município de São Caetano do Sul como professor da educação profissional técnica. Já em 2009, tornou-se corredor amador e, desde então, muitos problemas de TI são resolvidos durante os treinos de corrida.

Valéria Helena P. Gerbelli

Valéria é formada em nível técnico em Processamento de Dados em 1988, concluiu a graduação na área de Tecnologia em Processamento de Dados em 1992 e terminou a licenciatura plena em Informática em 1998. Formou-se em Pedagogia no ano de 2017.

Ela é pós-graduanda em Educação Profissional e Tecnológica e Docência em Ensino Superior. Ela também é professora de

Tecnologia da Informação desde 1995, e professora do Centro Estadual de Educação Tecnológica Paula Souza pelo Estado de São Paulo, nas ETECs, desde 1998. Para ela, ensinar e formar pessoas para o mundo tecnológico é muito importante, pois uniu sua formação na área de Tecnologia da Informação com a Educação.

PREFÁCIO

App Inventor 2 é um programa de computador de código aberto, no qual é possível desenvolver vários aplicativos para o sistema Android. Isso é realizado através de uma linguagem visual, não sendo necessário decorar ou escrever códigos de linguagem de programação. Com um simples arrastar e inserir blocos para a área de programação, o seu aplicativo será montado e desenvolvido.

Neste livro, os autores têm como objetivo proporcionar aos estudantes, educadores e entusiastas por tecnologia o primeiro passo no mundo da programação para dispositivos móveis por meio do App Inventor. Você aprenderá a desenvolver um aplicativo básico com alguns cálculos e outro até com um recurso de localização de GPS. Vamos criar também um tradutor de idiomas que possibilitará o compartilhamento da tradução com seus contatos do WhatsApp, e verá no projeto final como utilizar um aplicativo para se conectar a um banco de dados em MySQL, disponibilizando-o na loja do *Google Play Store*.

Ao tentarmos desenvolver aplicativos com o *Android Studio* em nossas aulas, deparamo-nos com a dificuldade de executar o ambiente de desenvolvimento, pois, para começar a trabalhar, isso tornava-se uma tarefa muito árdua, já que consumia boa parte do tempo das aulas. E para testar o que era realizado, também havia um grande tempo de espera, o que desmotivava muito os alunos, dispersando-os e fazendo-os perder o interesse pelo desenvolvimento de aplicativos.

Com a indicação e descoberta do MIT App Inventor 2,

encontramos um ambiente ideal para realizar nosso trabalho. Não existe praticamente nada de literatura no Brasil que fale sobre esse ambiente. Com um trabalho árduo de pesquisa e estudos, resolvemos escrever este livro, que é uma ferramenta interessante para aqueles que desejam ensinar e aprender a programar.

O livro poderá ser usado em qualquer nível da educação, da básica até a universitária, pois o aluno poderá desenvolver seu próprio aplicativo para estudo. Ele também auxilia os professores no processo de ensino-aprendizagem.

Pré-requisitos

A utilização das técnicas da lógica de programação não está descartada aqui, como não está descartada em nenhuma outra linguagem de programação. Mas mesmo sem esse conhecimento prévio, o leitor poderá acompanhar a leitura e o desenvolvimento dos aplicativos sugeridos neste livro.

O leitor deverá possuir um pequeno conhecimento da linguagem **PHP** para um melhor entendimento dos capítulos finais, porém, não é um fator limitante para um bom aprendizado.

O único pré-requisito é o desejo de aprender. Com isso, o leitor já poderá ser um inventor de aplicativos.

Público-alvo

Este livro foi escrito tanto para as pessoas que estão iniciando em programação para celulares e tablets quanto para aquelas que já possuem um conhecimento em programação. O público-alvo deste livro são pessoas de uma faixa etária entre 13 a 24 anos, porém,

qualquer um que queira aprender conseguirá se tornar um inventor de aplicativos.

O MIT App Inventor é uma ótima ferramenta de aprendizado. Qualquer pessoa que estiver disposta a ler este livro aprenderá o que é programação e se tornará um **inventor de aplicativos**.

Ele permite que os recém-chegados à programação de computador criem aplicativos de software para o sistema operacional Android. Ele usa uma interface gráfica que permite aos usuários arrastar e soltar objetos visuais para criar um aplicativo que pode ser executado em dispositivos desse sistema operacional.

Procuramos não utilizar termos técnicos da informática para tornar a leitura mais acessível e focar no desenvolvimento de aplicativos.

Sumário

1 Introdução **1**

1.1 Acessando o ambiente de desenvolvimento 2

1.2 Criando o primeiro app 4

1.3 Tour pelo ambiente de desenvolvimento 5

1.4 Inserindo um componente na tela 12

1.5 Resumindo 14

2 Executando o aplicativo **16**

2.1 Instalação do aplicativo 16

2.2 Emulando no computador 20

2.3 Resumindo 29

3 Uma simples calculadora **30**

3.1 Desenvolvendo o layout do aplicativo 32

3.2 Acessando a área de programação 55

3.3 Programando um botão 59

3.4 Testando o aplicativo 67

3.5 Resumindo 68

4 Etanol ou gasolina? **70**

4.1 Tela do aplicativo 70
4.2 Inserindo os blocos de controle 85
4.3 Tomando a decisão 88
4.4 Tela de informação 100
4.5 Testando o aplicativo 108
4.6 Resumindo 110

5 Tradutor online 112

5.1 App Tradutor 113
5.2 Blocos do tradutor 127
5.3 Reconhecimento de voz 128
5.4 Traduzindo o texto 131
5.5 Vocalização da tradução 133
5.6 Selecionando o idioma para a tradução 134
5.7 Compartilhando sua tradução 138
5.8 Testando o aplicativo 142
5.9 Resumindo 145

6 Onde estacionei? 146

6.1 Iniciando o layout do app 147
6.2 Identificando a latitude e a longitude do estacionamento 157
6.3 Armazenando a latitude e a longitude 163
6.4 Exibindo o endereço do estacionamento 169
6.5 Localizando o ponto de partida 170
6.6 Traçando a rota 173
6.7 Testando seu aplicativo 175
6.8 Resumindo 178

7 Configuração do servidor web **180**

7.1 Configurando o servidor web local 180

7.2 Localizando o número IP de seu computador 186

7.3 Reconhecendo o servidor pelo emulador 189

7.4 Resumindo 193

8 Criação do banco de dados com o MySQL **194**

8.1 O aplicativo 194

8.2 Conhecendo o phpMyAdmin 195

8.3 Definindo o banco de dados do aplicativo 198

8.4 Resumindo 204

9 Layout do app **205**

9.1 Definindo os componentes de tela 205

9.2 Arquivo gravar.php 216

9.3 Blocos para a inclusão 220

9.4 A lógica do botão Salvar 225

9.5 Recebendo e verificando os dados de resposta 231

9.6 Emulando para salvar as informações 233

9.7 Resumindo 237

10 Consultando o banco de dados **238**

10.1 Adicionando os componentes da consulta 238

10.2 Arquivo listar.php 240

10.3 Criando a lista de corridas 242

10.4 Exibindo a lista de corridas 248

10.5 Exibindo todos os campos da corrida 249

10.6 Emulando a consulta 262

10.7 Resumindo 264

11 Excluindo um registro 265

11.1 Arquivo excluir.php 266

11.2 Botão Excluir 268

11.3 Emulando a exclusão 271

11.4 Resumindo 273

12 Alterando um registro 274

12.1 Finalizando o layout 274

12.2 Arquivo alterar.php 275

12.3 A lógica do botão alterar 277

12.4 Programando a alteração 278

12.5 Emulando a alteração 285

12.6 Conectando com um banco de dados online 286

12.7 Resumindo 290

13 Publicando na Play Store 291

13.1 Criando uma conta de desenvolvedor 291

13.2 Informando os detalhes do app 294

13.3 Versões do app 297

13.4 Classificando o aplicativo 300

13.5 Preços e distribuição 304

14 Conclusão 310

Capítulo 1

INTRODUÇÃO

O App Inventor 2 é uma ferramenta inovadora, desenvolvida pelo Google e mantida hoje pelo Instituto de Tecnologia de Massachusetts (MIT). Ela cria aplicativos para dispositivos móveis, que rodam no sistema operacional Android, e transforma a complexa linguagem de programação baseada em texto em um ambiente baseado em blocos, totalmente visual — *drag and drop* (arraste e solte).

A interface é simples e funcional, porém, essa simplicidade não é sinônimo de criação de aplicativos básicos.

Recomendações básicas necessárias

Para utilização do App Inventor, o leitor deverá ter instalado um dos seguintes sistemas operacionais em seu computador:

- Windows — XP ou superior;
- PC (Macintosh com Processador Intel) — Mac OS X 10.5 ou superior;
- Linux — Ubuntu 8 ou superior, Debian 5 ou superior.

O leitor deverá usar um dos seguintes navegadores para o desenvolvimento de aplicativos utilizando o App Inventor:

- Google Chrome;
- Apple Safari;
- Mozilla Firefox.

Como funciona?

A Interface Gráfica com o Usuário (GUI — *Graphic User Interface*) permite criar a aparência visual de um aplicativo e, a partir dos eventos (ações) sobre os objetos, você "escreve" o programa juntando blocos que lembram um quebra-cabeça. Isso minimiza, e muito, a digitação de códigos, pois cada bloco terá sua funcionalidade bem definida.

O leitor não precisará instalar o App Inventor em seu computador, pois seu ambiente de desenvolvimento é realizado todo online, tornando-se necessário estar conectado à internet para usá-lo, como veremos a seguir.

1.1 ACESSANDO O AMBIENTE DE DESENVOLVIMENTO

Acesse o ambiente de desenvolvimento pelo link http://appinventor.mit.edu/explore/, e clique no botão `Create apps!` . A figura a seguir exibe o botão para acessar a área de criação de aplicativos.

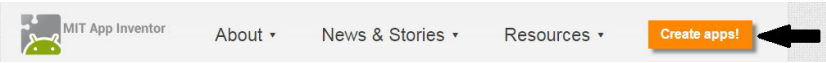

Figura 1.1: Acessando o ambiente de desenvolvimento

Ao clicar no botão, o Google nos informa que um aplicativo

solicita permissão para acessar sua conta e exibe seu endereço de e-mail. Clique no botão de permitir (`Allow`) para dar sequência e entrar no ambiente. A figura seguinte exibe a tela que solicita a permissão.

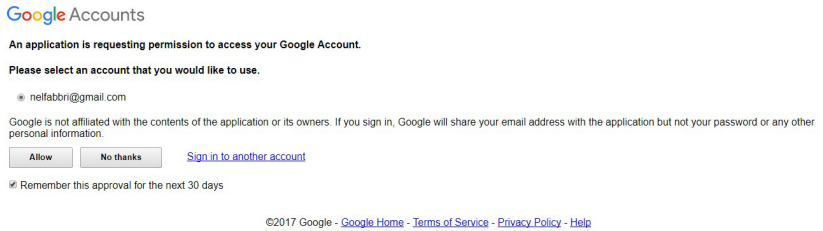
Google Accounts
An application is requesting permission to access your Google Account.
Please select an account that you would like to use.
nelfabbri@gmail.com
Google is not affiliated with the contents of the application or its owners. If you sign in, Google will share your email address with the application but not your password or any other personal information.
Allow
No thanks
Sign in to another account
Remember this approval for the next 30 days
©2017 Google - Google Home - Terms of Service - Privacy Policy - Help

Figura 1.2: Permissão para utilizar a conta do Google

Faça login com sua conta Google e clique no botão `Seguinte` . A próxima imagem exibe a tela para login.

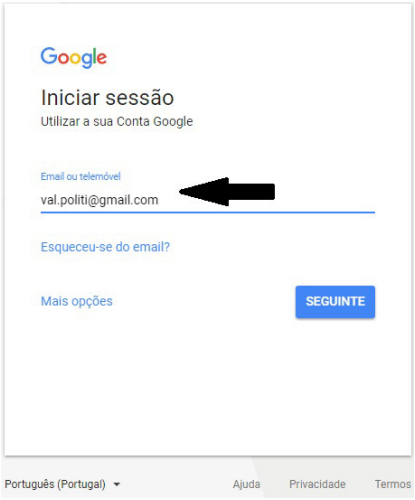

Figura 1.3: Tela de login no App Inventor

Surgirá uma nova tela para você inserir sua senha. Digite-a e

realize o login para acessar o ambiente de desenvolvimento do App Inventor. Após a liberação do uso com a senha, poderemos criar nosso primeiro projeto.

1.2 CRIANDO O PRIMEIRO APP

Ao se logar, a tela inicial do ambiente do App Inventor será exibida. Nela ficarão listados todos os seus projetos que ainda serão criados. A figura a seguir exibe a tela inicial.

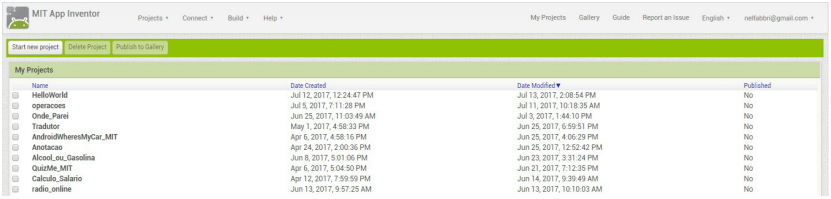

Figura 1.4: Tela de login no App Inventor

Para iniciar um novo projeto, clique no menu `Projects` e selecione a opção `Start new Project`, ou simplesmente clique diretamente no botão `Start new Project`. A próxima figura exibe a opção para criar um novo projeto.

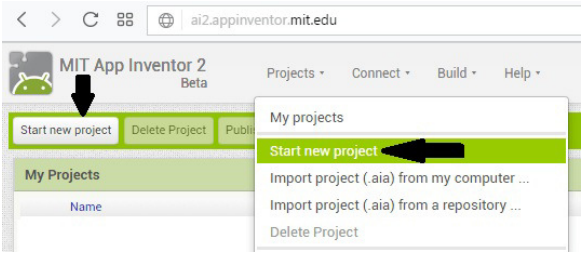

Figura 1.5: Iniciando um novo projeto

Será exibida uma tela para informar o nome do novo projeto,

conforme demonstra a figura a seguir. Dê o nome para seu projeto de **HelloWorld** e clique no botão `OK` para confirmar e continuar.

Figura 1.6: Nomeando um projeto

A maldição do Hello World!

Segundo a lenda, existe uma 'maldição' no mundo de TI que diz: se você está aprendendo uma nova linguagem de programação, você terá de fazer o primeiro programa para exibir na tela a mensagem **Olá Mundo**. Caso contrário, se não fizer, você não aprenderá essa nova linguagem. Você não vai se arriscar, vai?

1.3 TOUR PELO AMBIENTE DE DESENVOLVIMENTO

Antes de começar a desenvolver, você deve conhecer o ambiente com o qual vai trabalhar. Veja como é a tela do MIT App Inventor 2:

Figura 1.7: Conhecendo o ambiente de desenvolvimento

Pallete

É o local onde estão agrupados os componentes que serão utilizados para criação das *screens* (telas).

O QUE SÃO OS COMPONENTES?

São os elementos visuais que vão compor a tela de seu aplicativo e que darão interatividade com o usuário. Como exemplo, temos o botão e a caixa de texto para digitação de informações.

Eles são separados por diferentes guias, e estas indicam as categorias às quais os componentes pertencem.

Vejamos a seguir as guias de componentes que usaremos e suas

funcionalidades. Os componentes que serão usados nos aplicativos desenvolvidos no futuro serão explicados no momento de sua utilização.

- **Guia User Interface** — Essa guia contém todos os componentes que realizarão a interface com o usuário. É através desses componentes que o usuário poderá interagir com o aplicativo, e vice-versa.
- **Guia Layout** — Nesta guia, ficam os componentes que vão auxiliar a criar o layout do seu aplicativo, ajudando-nos a alinhar um ou mais componentes da guia User Interface em seu interior.
- **Guia Media** — Aqui estão os componentes que possibilitam o aplicativo a trabalhar com os dispositivos de áudio e vídeo de seu aparelho.
- **Guia Sensors** — Nela estão disponibilizados os componentes que possibilitam identificar os movimentos do seu celular ou tablet.
- **Guia Storage** — Ela nos apresenta os componentes de armazenamento de dados no dispositivo ou na internet.
- **Guia Connectivity** — Nela estão presentes os componentes que realizam a conectividade com outros aplicativos ou mesmo interagindo com a internet.

Área Viewer

Esta é a área onde você vai visualizar e inserir os componentes para criar o layout do seu aplicativo. Seus componentes ficarão

organizados da maneira como você definir e for alterando as suas propriedades. A figura a seguir exibe a área Viewer.

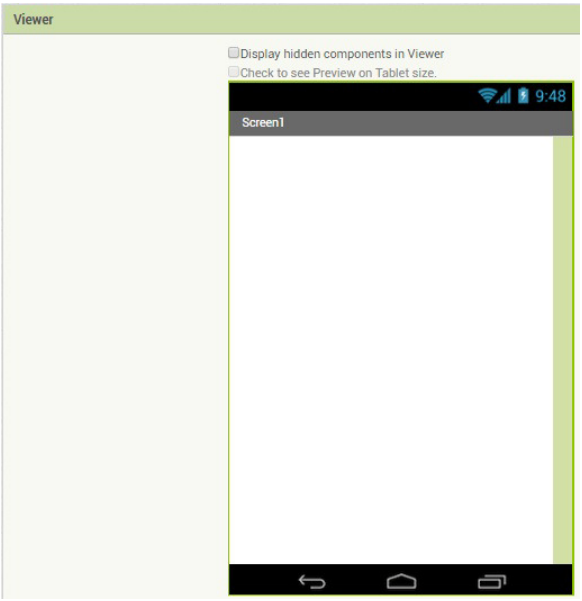

Figura 1.8: A área Viewer

Área Components

Nesta área, serão exibidos todos os componentes que estão sendo usados no projeto, dentro da área Viewer. Também é nela que podemos renomear os componentes e excluí-los, como veremos durante a criação do nosso primeiro aplicativo. A próxima figura exibe a área Components.

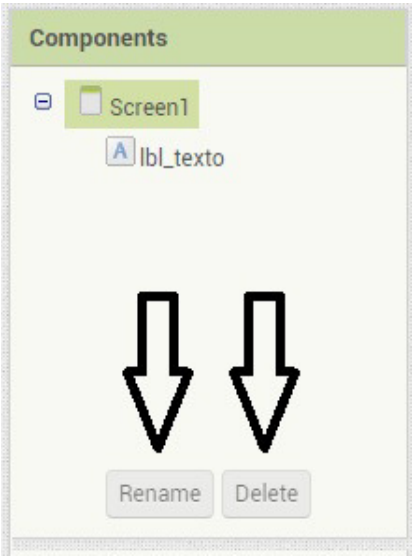

Figura 1.9: A área Components

Área Media

Nela, serão exibidos os arquivos que forem carregados para dentro do seu projeto, que podem ser de áudio, vídeo e imagens. Os arquivos suportados são:

- Áudio: `mp3` , `wav` , `mid` , `3gp` , `mp4` , `mkv` , `ota` , `imy` e `ogg` .
- Vídeo: `mp4` , `3gp` , `webm` e `mkv` .
- Imagens: `bmp` , `gif` , `jpg` , `png` e `webp` .

Área Properties

Permite alterar as características dos componentes que são inseridos na Viewer. Cada componente terá várias propriedades ou

configurações que poderão ser alteradas ou não. Tudo dependerá das necessidades que vão surgir no desenvolvimento do aplicativo.

Um exemplo seria um componente `Label` que tem a finalidade de exibir mensagens na tela do dispositivo, e ele poderá ter sua cor de fundo e tamanho da sua fonte modificados, assim como o texto a ser exibido. Essas modificações são chamadas de propriedades. A figura a seguir exibe a área das **Properties** de uma `Label` .

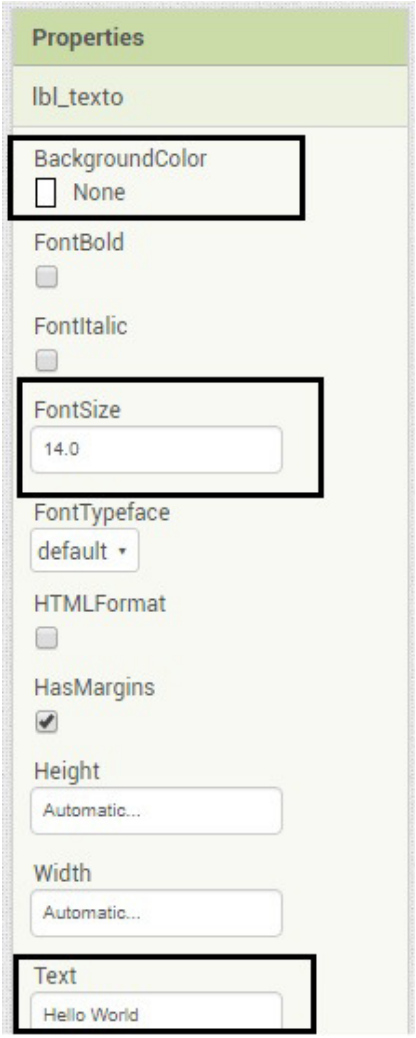

Figura 1.10: A área Properties

Essas propriedades também podem ser alteradas durante a execução do aplicativo, apenas deveremos saber o **nome do componente** e fazer referência à **propriedade** que desejamos

mudar — e não esquecer, claro, de indicar um novo valor.

1.4 INSERINDO UM COMPONENTE NA TELA

Após a realização da identificação das seções do ambiente de desenvolvimento do App Inventor, vamos retornar ao desenvolvimento do primeiro aplicativo, o **HelloWorld**. Já definimos e criamos o projeto, agora precisamos começar a montar a tela que o usuário verá ao executar o aplicativo.

Esse primeiro app apenas exibirá a mensagem **Hello World** na tela de seu dispositivo. Para a exibição de mensagens na tela para o usuário, utilizamos o componente `Label` .

Para inserir um componente que exiba um texto (ou seja, uma `Label`), clique na guia `User Interface` na área `Pallete` . Localize o componente, clique, arraste para a sua tela e solte. A figura a seguir demonstra esse procedimento.

Figura 1.11: Inserindo uma label

O componente ficará posicionado automaticamente na parte superior, alinhado à esquerda, conforme demonstrado na figura a seguir.

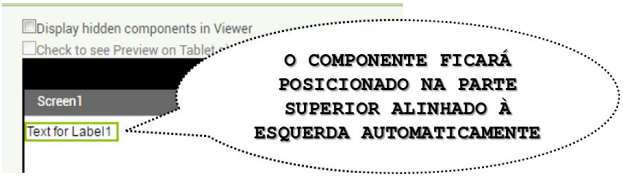
Display hidden components in Viewer
Check to see Preview on Tablet
Screen1
Text for Label1
O COMPONENTE FICARÁ POSICIONADO NA PARTE SUPERIOR ALINHADO À ESQUERDA AUTOMATICAMENTE

Figura 1.12: Label posicionada

Necessitamos alterar algumas configurações (propriedades) da `Label` que acabamos de inserir. É alterando as propriedades que podemos deixar o componente personalizado da maneira que queremos. Vamos alterar o texto de exibição e o tamanho da fonte.

Para isso, na área `Viewer`, clique sobre a `Label` para selecioná-la. Note que ela ficou selecionada com uma borda verde ao seu redor:

Figura 1.13: Label selecionada

Somente com o componente desejado selecionado é que poderemos alterar as suas propriedades. Vamos alterar o tamanho da fonte de 14 para 30, aumentando assim a sua visualização. Vá até a área das `Properties` e localize a propriedade `FontSize`. Logo abaixo dela, na caixa de texto, digite o número 30 e pressione a tecla `Enter` para a mudança ser assumida. A figura a seguir exibe a propriedade alterada e a visualização da `Viewer`.

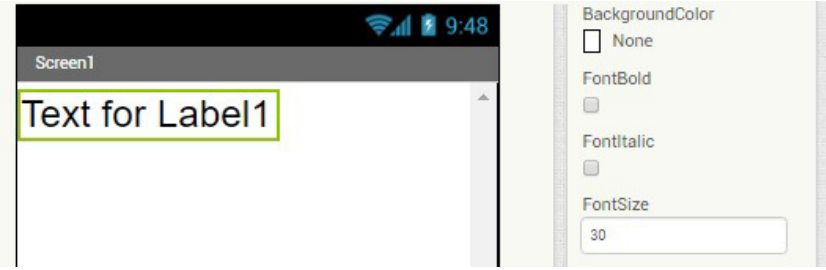

Figura 1.14: Propriedade fontsize alterada

Agora vamos alterar a propriedade que exibe o texto que desejamos. Com a `Label` ainda selecionada, vá novamente à área das `Properties` e localize a propriedade `Text` . Logo abaixo dela, digite o texto que será exibido na `Label` : **Hello World**. A figura a seguir exibe a tela finalizada do aplicativo.

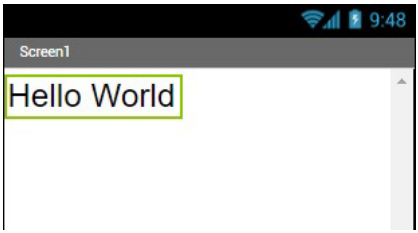

Figura 1.15: Tela finalizada

1.5 RESUMINDO

Finalizamos aqui esta introdução ao App Inventor. Realizamos um tour de reconhecimento do ambiente de seu desenvolvimento e criamos o layout do primeiro projeto. Porém, está faltando ainda realizarmos os testes para instalar e ver o seu funcionamento no celular ou tablet, e isso será visto no próximo capítulo. Até lá.

Realize o download de todos os arquivos utilizados nos projetos do livro em http://nelfabbri.com/appinventor/appinventor.zip.

CAPÍTULO 2

EXECUTANDO O APLICATIVO

No capítulo anterior, conhecemos o ambiente de desenvolvimento do App Inventor, desde como se logar até como criar um novo projeto. Fizemos um tour passando pelas áreas dos componentes e suas guias, e vimos a área de designer, de propriedades e de mídias. Também inserimos uma primeira `Label` e alteramos algumas de suas propriedades para preparar o primeiro aplicativo, o **HelloWorld**.

Esse simples aplicativo teve a função de apenas apresentar o App Inventor e os passos para um desenvolvimento. Após a criação do layout, necessitamos realizar os testes de funcionamento, executando-o. Para isso, existem três maneiras, que veremos a seguir.

2.1 INSTALAÇÃO DO APLICATIVO

A primeira maneira que veremos para testar seu aplicativo é instalando-o diretamente em seu celular ou tablet, com o sistema operacional Android.

Clique no menu suspenso `Build` do App Inventor, e selecione

a opção `App (provide QR code for .apk)`. A figura a seguir exibe a opção que gera um *QR code* para a instalação em seu dispositivo.

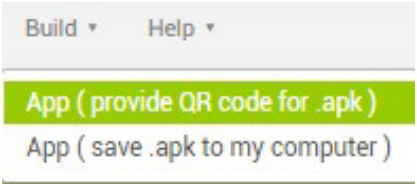

Figura 2.1: Gerar a instalação

Após o clique, começará a ser gerado um arquivo do tipo APK, e será exibida uma barra de progresso. Aguarde o processo chegar aos 100%.

> **APK** (Android Package) é um arquivo compilado, isto é, um pacote usado para instalar programas no Android.

Ao término da geração do pacote de instalação, a barra de progresso sumirá e será exibido um QR code. Esse QR code que acabou de ser gerado terá uma validade de, no máximo, duas horas para a instalação. Após esse período, caso queira uma nova instalação, deverá repetir o procedimento para gerar um novo.

Realize a leitura do QR code gerado na tela do computador, apontando a câmera de seu celular ou tablet. A figura a seguir exibe a realização da leitura pelo seu dispositivo. Clique no botão `Ligação aberta` para iniciar o download.

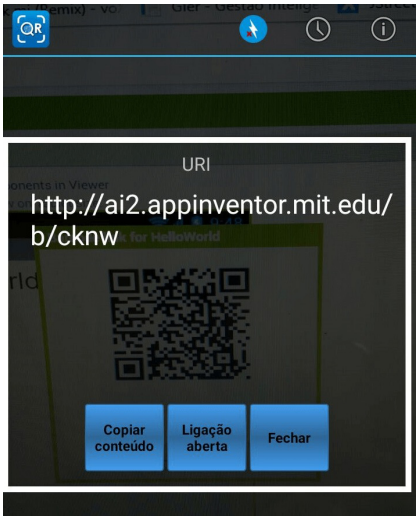

Figura 2.2: Gerar a instalação

> Se o seu tablet ou celular não possuir um leitor de QR code instalado, sugiro acessar a Play Store e realizar a instalação do Lightning QR, ou um similar.

Antes de iniciar o download, você será informado com a mensagem de alerta exibida na imagem a seguir. Essa informação surgirá porque não estamos baixando um arquivo da Play Store. Clique no botão `OK` para confirmar.

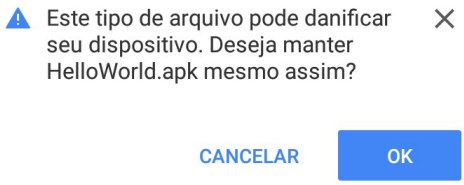

Figura 2.3: Download concluído

Após receber em seu dispositivo a informação de que o download foi concluído, clique no botão `Abrir` para iniciar a instalação. Para instalar o aplicativo em seu celular ou tablet, sua permissão será solicitada. Pressione o botão `Permitir` para iniciar.

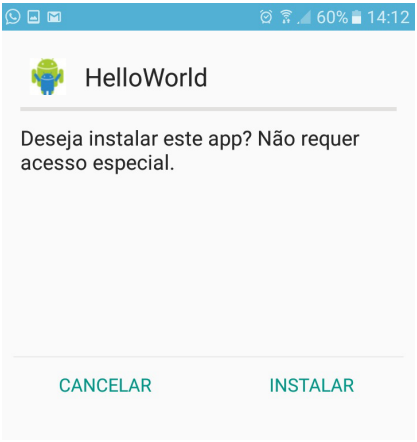

Figura 2.4: Permissão para instalação

Após a permissão, você verá a progressão em seu aparelho. Aguarde a conclusão da tarefa. Ao término da instalação, você receberá a informação de que o app foi instalado na tela de seu dispositivo.

Clique no botão `abrir` para executar seu app e veja o resultado conforme a figura:

Figura 2.5: App instalado

2.2 EMULANDO NO COMPUTADOR

Uma segunda maneira que temos para testar um aplicativo é utilizar o próprio emulador fornecido pelo App Inventor. Um emulador é um programa que simula o funcionamento de um dispositivo Android, mas que aparece na tela do seu computador. Chamamos o ato de testar seu app diretamente no ambiente de desenvolvimento de **emular seu aplicativo**.

Antes de emular seu aplicativo, você deve baixar o instalador do emulador. O **aiStarter** está disponível no site do App Inventor através do link http://appinventor.mit.edu/explore/ai2/setup-emulator.html. Realize o download conforme as orientações do site e de acordo com o sistema operacional instalado em seu computador.

Depois do download, clique sobre ele para executar e siga as instruções de instalação de acordo com o programa de *setup*. Após a instalação, um ícone do programa ficará disponível:

Figura 2.6: Ícone do aiStarter

Clique sobre o ícone para executá-lo. A tela demonstrada na imagem a seguir será exibida. Deixe-a em execução, não fechando a janela, pois ela é de fundamental importância para você emular seu aplicativo.

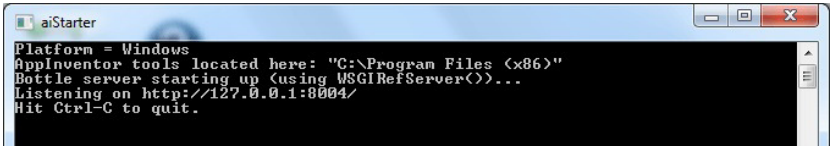
aiStarter
Platform = Windows
AppInventor tools located here: "C:\Program Files (x86)"
Bottle server starting up (using WSGIRefServer())...
Listening on http://127.0.0.1:8004/
Hit Ctrl-C to quit.

Figura 2.7: O aiStarter em execução

Depois de executado o aiStarter, vamos testar o nosso aplicativo diretamente no ambiente de desenvolvimento. Acesse o menu `Connect` e selecione a opção `Emulator` :

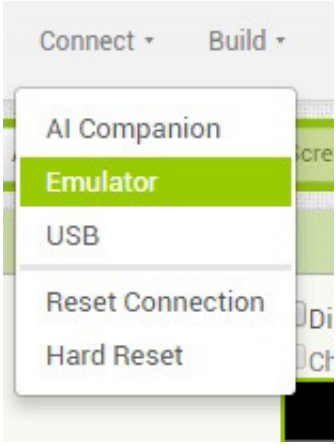

Figura 2.8: Emulando o seu projeto

A janela na próxima imagem será exibida, indicando que o emulador será iniciado. Aguarde.

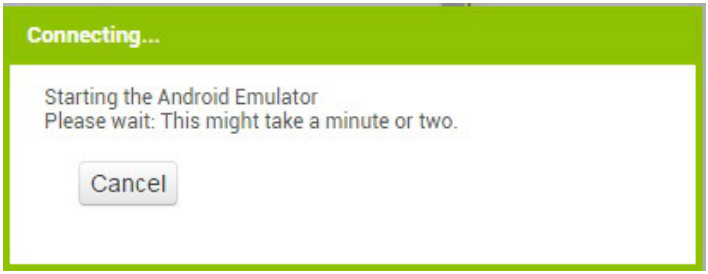

Figura 2.9: Emulador iniciando

Como é a primeira vez que estamos utilizando o emulador, o App Inventor provavelmente solicitará que façamos uma atualização. A figura seguinte exibe a mensagem solicitando um *update*. Caso a mensagem não apareça, ignore este procedimento.

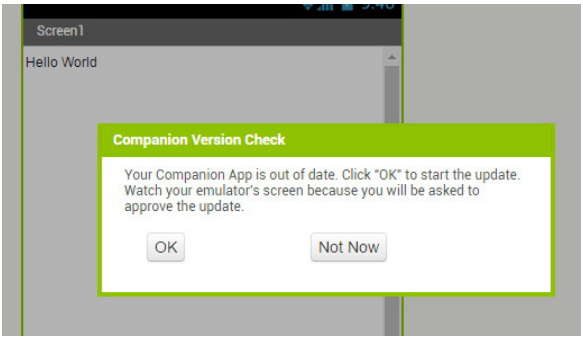
Screen1
Hello World
Companion Version Check
Your Companion App is out of date. Click "OK" to start the update. Watch your emulator's screen because you will be asked to approve the update.
OK
Not Now

Figura 2.10: Solicitação de update

Clique no botão `OK` para confirmar o *update*. Surgirá uma mensagem indicando que uma atualização está sendo feita em seu dispositivo (emulador), solicitando a sua permissão para a instalação. Clique em `Got it` para continuar. A figura a seguir exibe a tela com a mensagem de atualização.

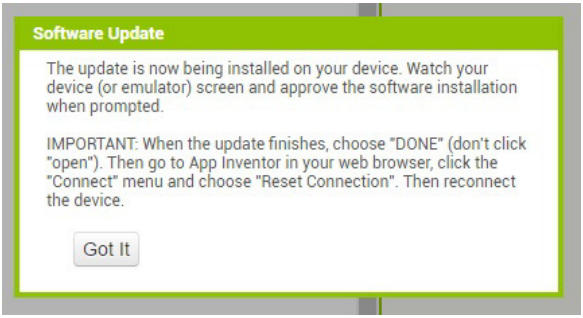
Software Update
The update is now being installed on your device. Watch your device (or emulator) screen and approve the software installation when prompted.
IMPORTANT: When the update finishes, choose "DONE" (don't click "open"). Then go to App Inventor in your web browser, click the "Connect" menu and choose "Reset Connection". Then reconnect the device.
Got It

Figura 2.11: Permissão para instalação

Após a sua permissão, perceba que na tela do emulador surgirá um aviso para atualização. Clique em `OK` para prosseguir.

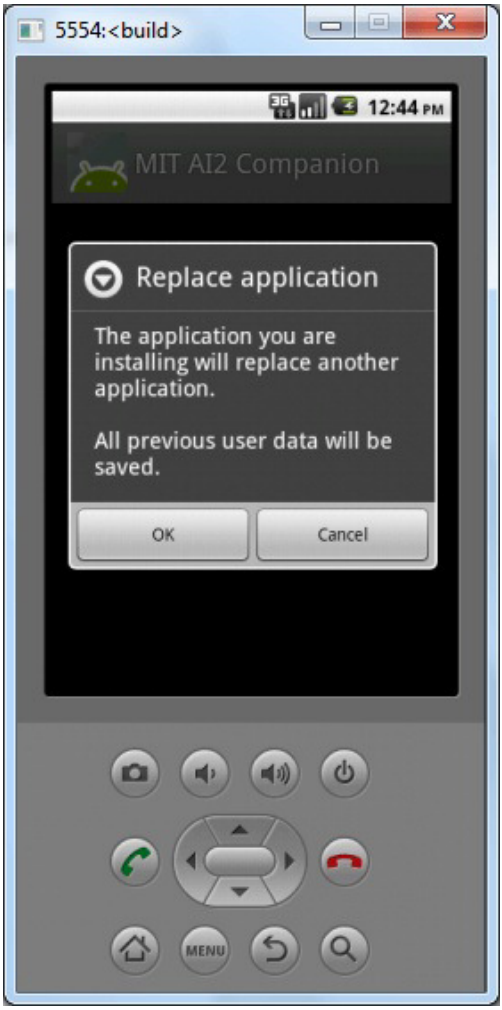

Figura 2.12: Atualização do emulador

A tela a seguir surgirá, solicitando permissão para instalar a atualização. Clique em `Install` .

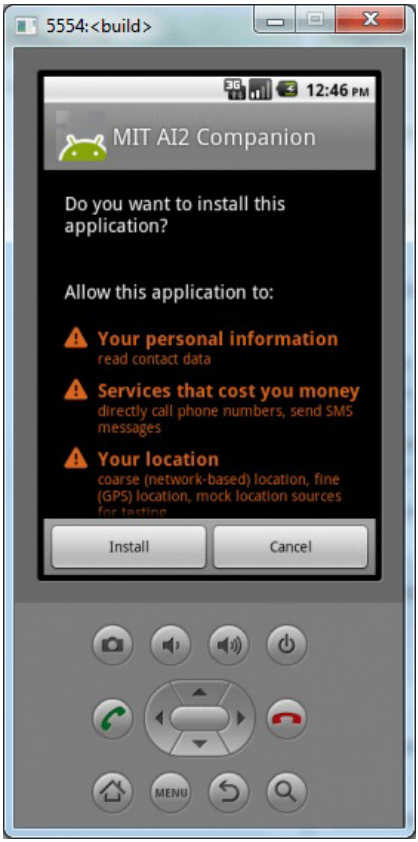

Figura 2.13: Permissão de atualização

Após sua permissão, você verá uma barra de progresso indicando a evolução da instalação e, ao término, surgirá a tela mostrada na imagem a seguir, indicando seu fim. Clique no botão `Open` para executar a sua aplicação.

Figura 2.14: Instalação finalizada

Caso o seu aplicativo não seja executado, clique novamente no menu `Connect` , selecione a opção `Emulator` e aguarde o carregamento do seu aplicativo.

Para as demais vezes que você quiser emular seu projeto, será necessário apenas ativar o *aiStarter*, depois clicar no menu `Connect` e selecionar a opção `Emulator` .

EMULANDO DIRETAMENTE EM SEU DISPOSITIVO

A terceira maneira que temos para emular o seu projeto é utilizar diretamente o celular ou o tablet, mas sem a necessidade de instalação como vimos antes. É uma maneira mais rápida e prática de testar seu app. Basta conectar o computador via cabo USB com o dispositivo, porém será necessário que você tenha acesso a uma rede Wi-Fi e o aplicativo aiStarter instalado em seu aparelho.

Entre na Play Store através do seu dispositivo, localize e instale o aplicativo **aiStarter**. Após a instalação, para emular o aplicativo, abra o **MIT AI2 Companion** em seu celular ou tablet. A figura a seguir exibe a tela do aplicativo inicializado.

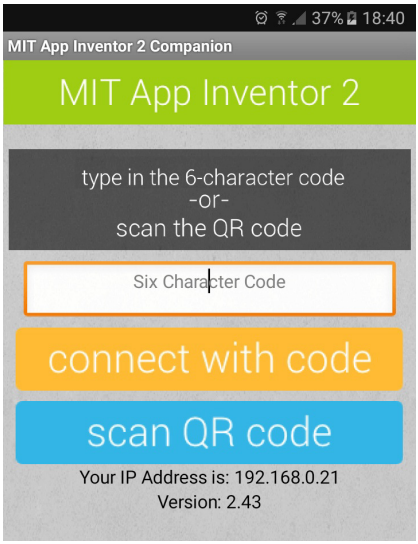

Figura 2.15: aiStarter em execução no dispositivo móvel

Para conectar o App Inventor em seu dispositivo, acesse o menu `Connect` e selecione a opção `All Companion` :

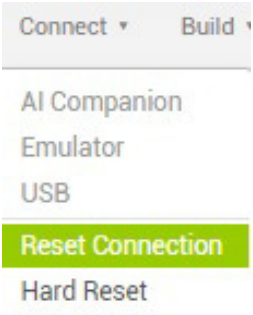

Figura 2.16: Conectando com o dispositivo

Note que foi gerado um QR code e também um código com letras e números que servirá para ativar a emulação. Volte para o celular ou tablet e, na caixa de texto do aplicativo **aiStarter**, digite

o código que foi gerado na tela do computador através do QR code. Na sequência, clique no botão `Connect with code` .

O celular deverá exibir seu projeto funcionando. Mas lembre-se de que você está emulando e não instalou em seu dispositivo, diferentemente da primeira maneira apresentada no início deste capítulo. Ao desconectar o celular ou fechar o ambiente de desenvolvimento, ele não estará mais disponível no seu dispositivo móvel.

2.3 RESUMINDO

Neste capítulo, conhecemos as três maneiras de testar nosso aplicativo: instalando, emulando no computador e emulando no próprio dispositivo com Android. No próximo, veremos como adicionar outros componentes da guia `User Interface` , e começaremos a conhecer como inserir interatividade com os componentes.

Capítulo 3

UMA SIMPLES CALCULADORA

Após ter realizado um tour de reconhecimento do ambiente do App Inventor, aprenderemos neste capítulo a declarar variáveis, utilizar botões para controlar a solicitação do usuário, realizar pequenos cálculos e exibir o resultado.

Passo a passo, você verá a explicação e a demonstração de como criar uma simples calculadora que realizará as quatro operações básicas da matemática – soma, subtração, multiplicação e divisão. Apesar de ser um aplicativo bem simples, servirá para demonstrar como utilizar algumas das seções do ambiente do App Inventor, apresentadas anteriormente, que usaremos nos próximos capítulos.

Iniciaremos criando um novo projeto. Para isso, clique no menu superior do App Inventor, na opção `Projects` , e depois em `Start new Project` . A figura a seguir exibe o local para a criação de um novo projeto. Sempre que desejarmos criar um novo projeto, será através deste caminho.

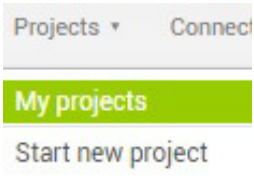

Figura 3.1: Criando um novo projeto

Após o clique, surgirá a tela `Create new App Inventor Project` , conforme a figura a seguir, onde daremos um nome para o projeto que será desenvolvido. Digite a palavra `Operacoes` em `Project Name` e, na sequência, confirme clicando no botão `OK` .

O nome `Operacoes` foi escrito sem acentos propositalmente, pois o App Inventor não permite a utilização de letras com acentuação, espaços em branco ou caracteres especiais.

> Baixe as imagens que serão utilizadas neste projeto, conforme informado no capítulo *Introdução*, em http://nelfabbri.com/appinventor/appinventor.zip.

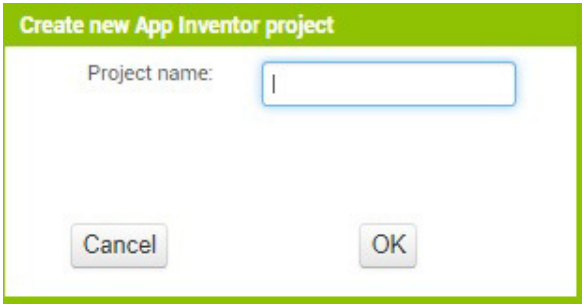

Figura 3.2: Criando um novo projeto

Surgirá na sua tela a `Screen1` , local no qual colocaremos os componentes visuais de nosso aplicativo. Esses componentes servirão para dar interatividade com o usuário, e aos poucos vamos discutindo sobre eles. A figura a seguir exibe a `Screen1` pronta para a inserção dos componentes.

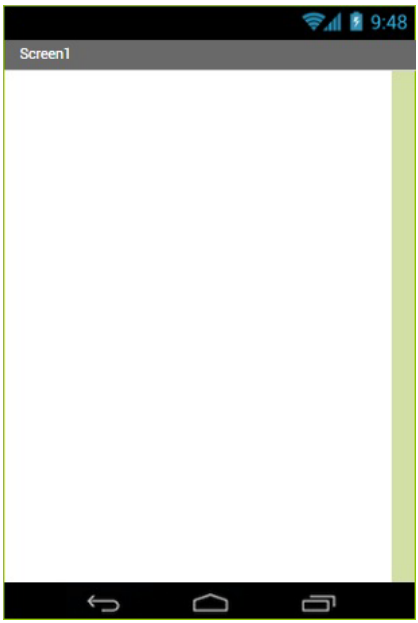

Figura 3.3: Tela do projeto em branco

3.1 DESENVOLVENDO O LAYOUT DO APLICATIVO

Antes de inserir as funções que darão interatividade ao nosso aplicativo, necessitamos desenvolver o seu layout.

Quando usamos o termo *layout*, estamos nos referindo ao visual que nosso aplicativo terá, ou seja, onde ficarão disponibilizados os botões, as caixas de textos, as imagens e todos os outros componentes que poderemos utilizar na criação da aparência do app.

Começaremos a inserir uma imagem de fundo na `Screen1`. Necessitamos realizar um upload da imagem, ou seja, enviar a que desejamos exibir para o ambiente do App Inventor. Selecione a propriedade `BackgroundImage` na barra `Properties`, clicando no espaço imediatamente inferior à propriedade, para exibir a opção para enviar a sua imagem.

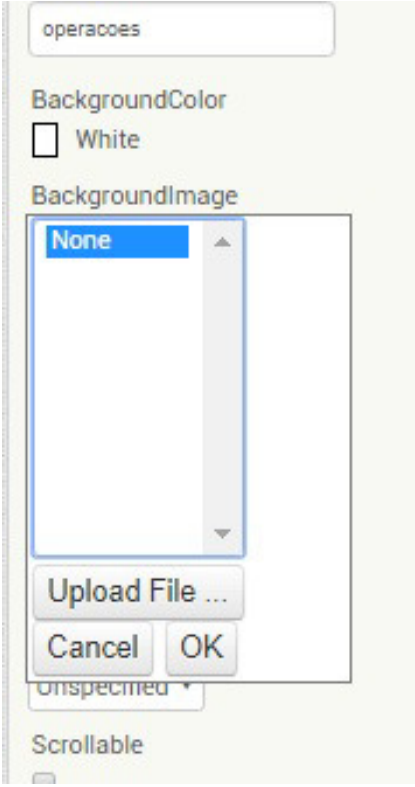

Figura 3.4: Propriedade BackgroundImage

Para enviar um arquivo, clique no botão `Upload File...`, assim será exibida a janela onde você escolherá a imagem para o envio. Utilize o arquivo `quadro.png`, que você já deve ter baixado, ou outra imagem qualquer de sua livre escolha. Sugiro usar imagens com o formato `.png`, pois este tem uma melhor qualidade de compressão de arquivos, sem perder a qualidade nem as suas cores.

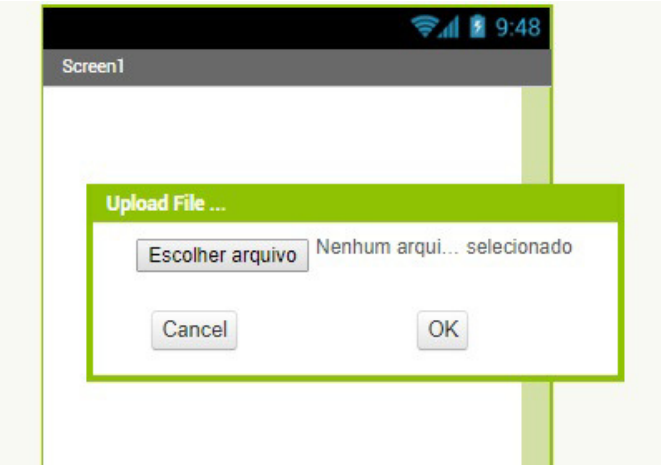

Figura 3.5: Selecionando o arquivo para upload

Abra o local onde você baixou os arquivos do projeto e escolha o `quadro.png`, então clique no botão `OK` para finalizar o envio. Aguarde o carregamento e, ao término, a sua `Screen1` deverá ter tal aparência:

Figura 3.6: Screen1 após o envio da imagem

Vamos alterar algumas outras propriedades da `Screen1` . O nome que demos no momento da criação do projeto, `Operacoes` , não é necessariamente o que será exibido embaixo do ícone após a instalação no dispositivo. O nome de exibição pode ser alterado, e é exatamente isso que vamos fazer agora.

Localize na barra de `Properties` a opção `AppName` e digite: **Operações**. Nesse local, podemos inserir textos com acentuação, espaços e qualquer outro tipo de caractere.

Uma outra mudança que realizaremos na barra de propriedades da `Screen1` é o nome que será exibido na barra de título do aplicativo. Observe na imagem a seguir como está essa exibição antes da mudança.

Figura 3.7: Barra de título original

Para alterar o nome da barra de título, vá à barra de `Properties` , localize a opção `Title` e altere o texto de `Screen1` para **As 4 operações**, pois este será o novo nome que exibiremos na barra de título de nosso app. A imagem a seguir exibe a tela de desenvolvimento com essa alteração.

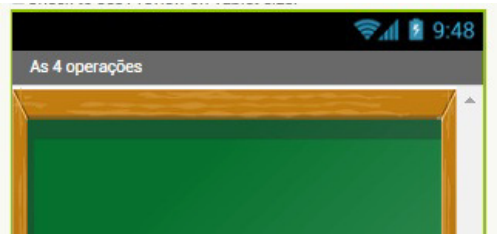

Figura 3.8: Barra de título modificada

> **DICA**
>
> Não é necessário modificar todas as propriedades de cada componente, alteraremos somente aquelas necessárias para cada aplicativo desenvolvido neste livro.

Dando sequência ao desenvolvimento do layout de nossa calculadora, inseriremos os componentes para montar o visual do

app. Vamos colocar uma mensagem na tela informando o que o aplicativo será capaz de realizar: **AS 4 OPERAÇÕES**.

> Ao inserir o primeiro componente, ele sempre será posicionado na parte superior esquerda da sua `Screen1` .

Para essa mensagem não se sobrepor a margem da lousa da imagem usada como fundo da `Screen1` , necessitamos de um componente para organizar a visualização da mensagem. Os componentes da guia `Layout` têm a finalidade de apenas organizar o designer dos demais componentes.

Vamos inserir um componente que realiza essa organização. Para isso, clique na guia `Layout` para exibir todos os componentes referentes à organização:

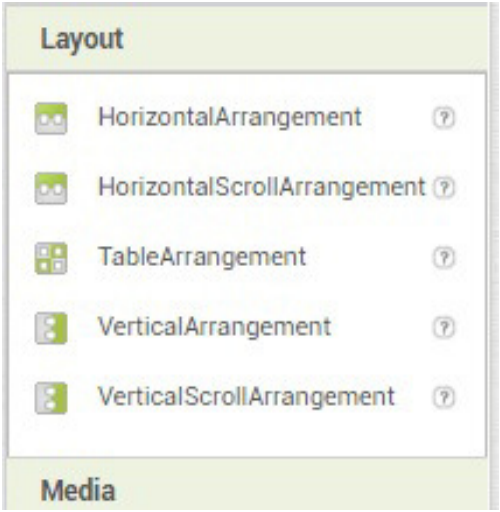

Figura 3.9: Guia Layout e seus componentes

O primeiro componente que usaremos é o `HorizontalArrangement`. Ele é capaz de organizar horizontalmente um ou mais componentes em seu interior. Sem ele, não conseguiríamos colocar dois componentes um ao lado do outro.

Para inserir um componente na `Screen1`, clique sobre ele, mantenha pressionado e arraste-o para a área da `Screen1`. Veja na figura a seguir como ela ficará após a inserção do `HorizontalArrangement`. Não se preocupe com o resultado deste componente, pois vamos alterar as suas propriedades e você verá como ele ficará.

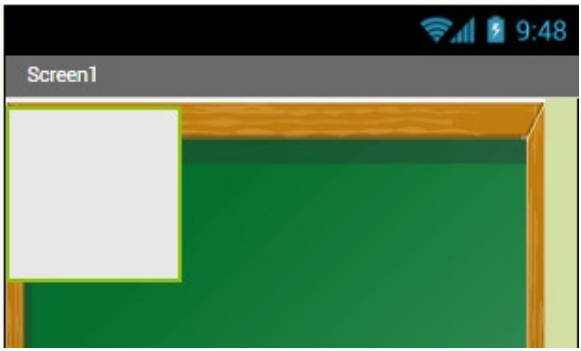

Figura 3.10: HorizontalArrangement

Altere as propriedades do componente `HorizontalArrangement`, conforme a tabela:

Propriedade	Alterar valor para	Observação
AlignVertical	Bottom:3	Indica que todos os componentes que estarão dentro do HorizontalArrangement serão alinhados na sua parte inferior.

BackgroundColor	None	Altera a cor de fundo do componente para transparente.
Height	80 pixels	Ajusta a altura do componente em exatos 80 pixels.
Width	Fill Parent	Ajusta o componente para a largura máxima da tela do dispositivo.

Utilizamos a propriedade `AlignVertical` para alinhar o texto na parte inferior da `HorizontalArrangement`, pois assim o texto que ainda vamos inserir ficará abaixo da margem da lousa que utilizamos de fundo. Alteramos para transparente a propriedade `BackgroundColor`, para que a imagem de fundo também seja exibida dentro do `HorizontalArrangement`.

Em `Height`, alteramos a altura do componente para `80 pixels` para forçar a exibição do texto abaixo da margem da lousa. E na propriedade `Width`, para o componente ocupar toda a largura da tela, selecionamos a opção `Fill Parent`.

Veja na figura a seguir como ficará a sua tela após a configuração das propriedades do `HorizontalArrangement`.

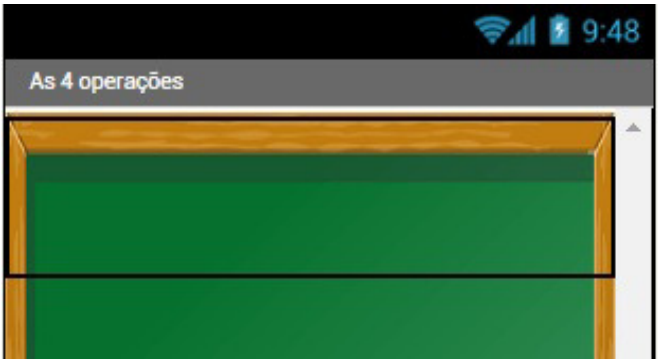

Figura 3.11: HorizontalArrangement configurado

Dentro do `HorizontalArrangement`, queremos exibir a mensagem **As 4 operações**, que será o título do nosso aplicativo. Existe um componente específico para essa exibição, a `Label`. Vamos então inseri-la em nossa `Screen1`.

Clique na `Pallete`, e depois na opção `User Interface` para exibir todos os componentes com que o usuário poderá interagir. A figura a seguir exibe a guia `User Interface` expandida.

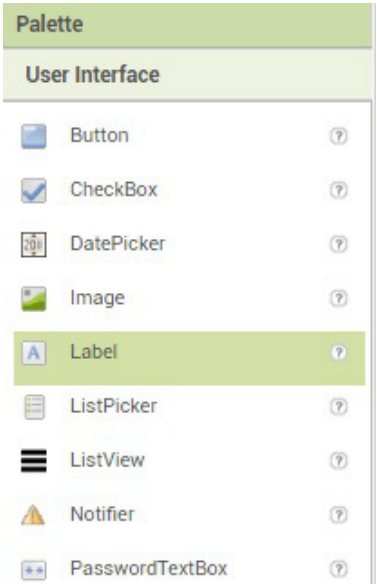

Figura 3.12: Guia User Interface

Selecione o componente `Label`, clique sobre ele e arraste para dentro do componente `HorizontalArrangement` na sua `Screen1`. A figura a seguir exibe como deverá ficar a sua tela de designer.

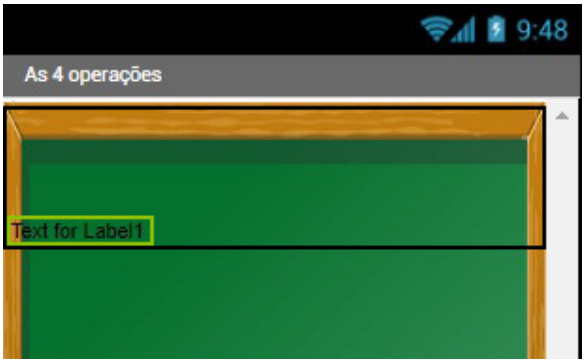

Figura 3.13: Tela com a Label inserida

Altere as propriedades do componente `Label`, conforme orientações da tabela:

Propriedade	Alterar valor para	Observação
FontBold	Marcar	Ativa a opção de negrito.
FontSize	25	Altera o tamanho da fonte.
Width	Fill Parent	Ajusta o componente para a largura máxima da tela do dispositivo.
Text	AS 4 OPERAÇÕES	Será o texto que será exibido na Label.
TextAlignment	center:1	Alinhamento do texto dentro da label.
TextColor	White	Cor de exibição do texto da label.

Como queremos dar um destaque ao texto **AS 4 OPERAÇÕES** que o leitor já deve ter digitado na propriedade `Text`, alteramos a que o deixa em negrito (`FontBold`), e também mudamos o tamanho da fonte através da propriedade `FontSize`, que foi modificada para o tamanho 25. Como desejamos que a `Label` ocupe todo o espaço da linha em que se encontra, alteramos a

propriedade `Width` para `Fill Parent` .

Para centralizar o texto em todo o espaço da `Label` , a propriedade `TextAlign` foi alterada para a opção `center:1` . E como a imagem de fundo do app é uma lousa verde, achamos interessante alterar a cor da `Label` para branco e, por essa razão, na propriedade `TextColor` , selecionamos `white` .

Após as configurações citadas, sua `Screen1` deverá ter a aparência:

Figura 3.14: Label configurada

Continuando o desenvolvimento do layout, necessitamos preparar o espaço para o usuário digitar um número. Como teremos na mesma linha uma mensagem indicando o que se deve digitar e uma caixa de texto para a digitação do número, obrigatoriamente teremos de inserir mais um `HorizontalArrangement` da guia `Layout` . Insira-o abaixo do `HorizontalArrangement` que contém a informação **AS 4 OPERAÇÕES** e vamos realizar as configurações conforme a tabela seguinte:

	Alterar	

Propriedade	valor para	Observação
AlignHorizontal	Center:3	Indica que todos os componentes que estarão dentro do HorizontalArrangement serão alinhados ao centro.
AlignVertical	Center:2	Indica que todos os componentes que estarão dentro do HorizontalArrangement serão alinhados ao centro.
BackgroundColor	None	Altera a cor de fundo do componente para transparente.
Width	Fill Parent	Ajusta o componente para a largura máxima da tela do dispositivo.

Conforme vimos anteriormente, a propriedade `AlignHorizontal` é utilizada para alinhar os componentes em seu interior e, como agora queremos deixá-los centralizados, usamos a opção `Center:3` . A propriedade `AlignVertical` alinha os componentes de maneira vertical, e então selecionamos a opção `Center:2` .

Alteramos a propriedade `BackgroundColor` para transparente pelo mesmo motivo da anterior, ou seja, para que possa ser exibida a imagem de fundo também dentro do `HorizontalArrangement` . Com a propriedade `Width` , selecionamos a opção `Fill Parent` para que o componente ocupe toda a largura da tela.

O usuário deverá informar um número para o aplicativo poder realizar os cálculos. O objeto que recebe a digitação de valores é o componente `TextBox` . Volte para a guia `User Interface` da `Pallete` e insira uma `Label` e uma `TextBox` no interior da `HorizontalArrangement` que acabamos de configurar.

Vamos configurar as seguintes propriedades desta forma:

Componente	Propriedade	Alterar valor para	Observação
Label	FontBold	Marcar	Ativa a opção de negrito.
	FontSize	16	Altera o tamanho da fonte.
	Text	Número	O texto que será exibido na Label.
	TextColor	White	Cor de exibição do texto da Label.
TextBox	Hint	Apagar o texto	Exibe um texto antes da digitação na TextBox.
	NumbersOnly	Marcar	Quando marcada, ativa o teclado numérico do seu dispositivo.

Para a `Label`, alteramos o texto de exibição para **Número**, pois assim o usuário do app saberá o que deverá digitar. Deixamos o texto em negrito e alteramos o tamanho da fonte para 16, utilizando respectivamente as propriedades `FontBold` e `FontSize`. Alteramos também a cor na propriedade `TextColor` para `White` para seguir com a mesma cor já utilizada anteriormente.

Para o componente `TextBox` que receberá o número digitado pelo usuário, apagamos os valores da propriedade `Hint`, pois não é necessária a exibição de nenhuma informação adicional, já que temos uma `Label` que indica o que deverá ser digitado em seu interior. Como o esperado na digitação é apenas números, devemos marcar a opção `NumbersOnly`, pois ela ativa um teclado que exibe apenas números durante a digitação, evitando assim que o usuário digite letras ou outros valores que não sejam números e que causariam erros na execução do app.

Veja na imagem a seguir a `Screen1` após as configurações realizadas na `Label` e na `TextBox`.

Figura 3.15: Label e TextBox configuradas

Para facilitar a identificação dos componentes, vamos renomear os que usaremos durante a programação. Em vez de chamarmos de `Textbox1`, trocaremos o seu nome para `Txt_N1`, fazendo uma referência ao fato de essa `TextBox` representar o primeiro número.

Para renomear, clique sobre o nome `Textbox1` na área `Components` para selecioná-lo e, logo após, clique sobre o botão `Rename`. A figura a seguir exibe o botão para renomear um componente.

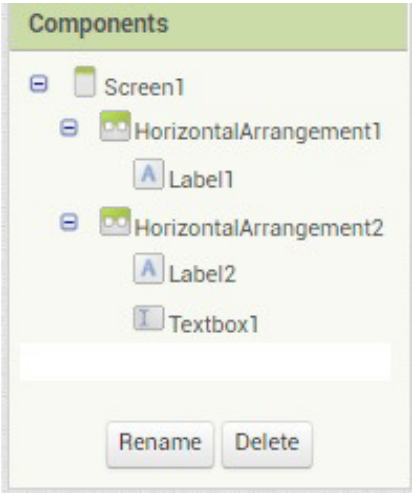

Figura 3.16: Área para renomear componentes

Após o clique, a janela `Rename Component` será exibida, conforme vemos a seguir. Altere a opção `New Name` para o novo nome que daremos ao componente `Txt_N1`, e clique no botão `OK` para confirmar.

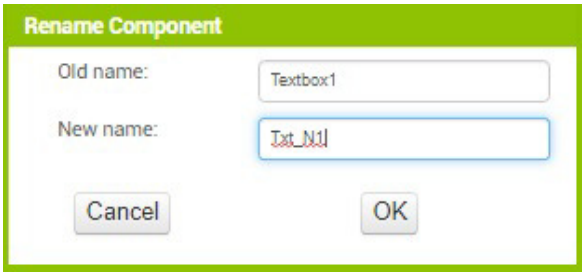

Figura 3.17: Renomeando o componente

Repita o procedimento visto anteriormente, inserindo um novo `HorizontalArrangement`, uma `Label` e outra `TextBox`, para permitir a digitação de um segundo número, não se

esquecendo de renomear a sua `TextBox` para `Txt_N2` . A figura seguinte exibe como deverá ficar o layout de seu app após as configurações realizadas.

Figura 3.18: Layout com as duas TextBox

Para dar um espaçamento entre os componentes já inseridos e os botões das operações matemáticas que adicionaremos mais adiante, precisamos utilizar um novo `HorizontalArrangement` . Este deverá ser inserido abaixo do `HorizontalArrangement` que contém o segundo número para a digitação.

Como sua única função é realizar essa separação, ele deverá ficará vazio sem nenhum outro componente em seu interior. Porém teremos de alterar algumas de suas propriedades: `Height` para `40` pixels, pois assim será inserido um bom espaçamento para o layout do app. Na propriedade `Width` , altere para `Fill Parent` .

A figura a seguir demonstra como ficará sua tela até este momento:

Figura 3.19: HorizontalArrangement servindo de espaçador

Precisamos de mais um `HorizontalArrangement` para organizar os quatro botões que realizarão as operações matemáticas de adição, subtração, multiplicação e divisão. Selecione na `Pallete` a guia `Layout` e arraste-a para a área de designer do aplicativo logo abaixo do `HorizontalArrangement` que utilizamos para realizar um espaçamento entre os componentes. Realize as seguintes alterações nas suas propriedades, conforme a tabela a seguir:

Propriedade	Alterar valor para
AlignHorizontal	Center:3
Width	Fill Parent

Como gostaríamos de centralizar nessa linha todos os futuros componentes que vamos inserir lado a lado em seu interior,

alteramos a propriedade `AlignHorizontal` para `Center:3` . Novamente na propriedade `Width` , foi selecionada a opção que deixa o componente utilizando todo o tamanho da linha: `Fill Parent` .

Acesse a `Pallete` e, na guia `User Interface` , selecione o componente `Button` e arraste-o para dentro do `HorizontalArrangement` . Esse primeiro componente `Button` , quando pressionado, executará as ações para realizar a adição dos números que vamos programar ainda neste capítulo.

Faça as seguintes configurações no botão, conforme a tabela:

Propriedade	Alterar valor para
FontSize	30
Height	50 pixels
Width	40 pixels
Text	+

Neste componente `Button` , foi alterado o tamanho da fonte para 30 na propriedade `FontSize` , para dar uma maior visibilidade ao comando que ele executará quando pressionado. Alteramos também o tamanho do botão para que fique com um layout mais proporcional ao restante do app. Para isso, na propriedade `Height` que altera a altura do botão, escrevemos `50` pixels e, para a sua largura (`Width`), escrevemos `40` pixels. E por fim, alteramos o texto de exibição do `Button` para `+` na propriedade `Text` .

Renomeie o botão para `Btn_Soma` conforme vimos anteriormente. Insira mais três botões ao lado direito do

`Btn_Soma` , ainda dentro da `HorizontalArrangement` , para realizar a subtração, multiplicação e divisão, e realize as configurações descritas na tabela a seguir:

Componente	Propriedade	Alterar valor para
Button2	FontSize	30
	Height	50 pixels
	Width	40 pixels
	Text	-
	Rename	Btn_Sub
Button3	FontSize	30
	Height	50 pixels
	Width	40 pixels
	Text	X
	Rename	Btn_Mult
Button4	FontSize	30
	Height	50 pixels
	Width	40 pixels
	Text	/
	Rename	Btn_Div

Da mesma maneira e com os mesmos objetivos que realizamos as configurações com o `Btn_Soma` , deveremos alterar as propriedades dos botões de **subtração**, **multiplicação** e **divisão** descritos na tabela anterior.

Veja na figura a seguir como ficará o design após a inserção e configuração dos botões.

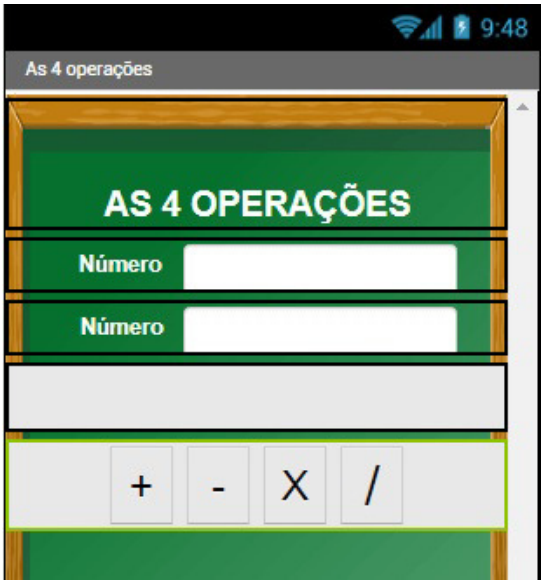

Figura 3.20: Exibição dos botões de cálculo

Finalmente, necessitamos de um espaço para exibirmos o resultado da conta escolhida pelo usuário. Como dissemos no começo deste capítulo, o componente responsável pela exibição de valores é o `Label`, porém, para a sua organização visual, devemos inserir mais um `HorizontalArrangement` logo abaixo do `HorizontalArrangement` que contém os botões com as operações matemáticas, e vamos alterar as suas propriedades, conforme a tabela a seguir.

Propriedade	Alterar valor para
AlignHorizontal	Center:3
AlignVertical	Center:2
Height	50 pixels
Width	Fill Parent

Alteramos as propriedades `AlignHorizontal` e `AlignVertical` para `Center`, para deixar os componentes que ficarão em seu interior centralizados. Deixamos também, por uma questão de layout, a sua altura em `50` pixels, alterando a propriedade `Height`. Já na propriedade `Width`, alteramos a sua largura para o comprimento total, selecionando a opção `Fill Parent`.

Dentro deste último `HorizontalArrangement`, insira duas `Labels`, em que a primeira exibirá a mensagem `Resultado` e a segunda exibirá o resultado do cálculo obtido. Veja na tabela a seguir as propriedades que deveremos alterar.

Componente	Propriedade	Alterar valor para
Label4	FontSize	20
	Text	Resultado:
Label5	FontSize	20
	Text	0
	Rename	Lbl_Resultado

Na primeira `Label`, que exibirá a mensagem que foi alterada na propriedade `Text` para **Resultado**, alteramos o tamanho da sua fonte para 20. Ela é apenas informativa, ou seja, somente exibirá a palavra **Resultado**.

A segunda `Label` que foi inserida servirá para exibir o resultado do cálculo matemático. Alteramos também o tamanho da fonte para 20 em `FontSize`, e definimos a propriedade `Text` para o valor **0**, pois assim que o aplicativo for inicializado, o número **0** já estará sendo exibido na tela do app. Renomeamos essa `Label` para `Lbl_Resultado` para uma melhor identificação

desse componente no momento em que estivermos programando as funções de cada botão e quisermos exibir o resultado do cálculo.

A figura a seguir exibe a tela final do aplicativo:

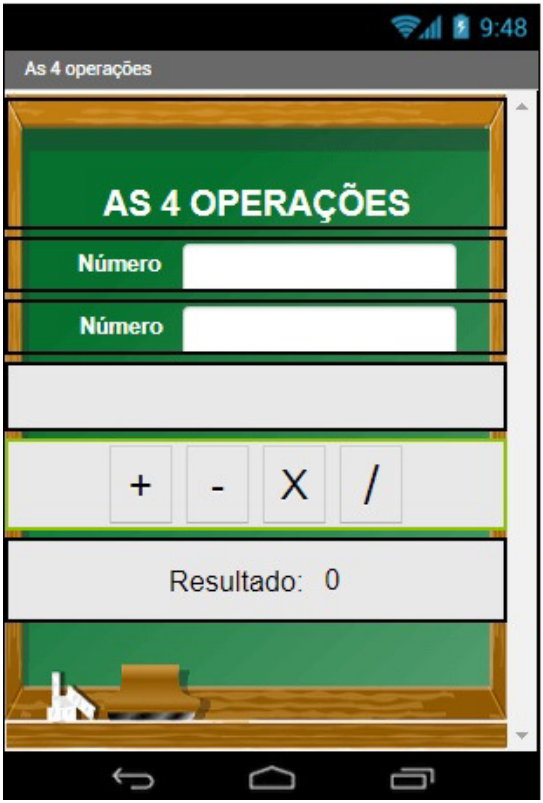

Figura 3.21: Layout completo do aplicativo

Nesse momento, finalizamos a criação do layout, já inserimos todos os componentes e configuramos as propriedades necessárias. Nosso próximo passo será preparar a interatividade do usuário com o aplicativo.

3.2 ACESSANDO A ÁREA DE PROGRAMAÇÃO

A área de programação, ou simplesmente a área dos blocos, é o local onde faremos a atribuição das funções para cada um dos componentes. Assim como existem várias propriedades e não modificamos todas, ocorrerá o mesmo com os componentes, pois vamos atribuir as funções, isto é, programar apenas os blocos necessários.

Para acessar a área de blocos, clique no botão `Blocks` no canto superior direito. A figura a seguir exibe o botão de acesso à área de blocos.

Figura 3.22: Acessando a área de blocos

Os campos `Txt_N1` e `Txt_N2` são os locais onde o usuário vai digitar os números para efetuarmos os cálculos. Logo, é necessário armazenar os valores digitados nessas `TextBoxs` em variáveis.

Começaremos a criar as variáveis que vamos usar no aplicativo. Precisaremos de duas variáveis para armazenar os valores digitados nas `TextBoxs` : `Txt_N1` e `Txt_N2` . Vamos conhecer qual é o procedimento para isso.

Do lado esquerdo da tela do App Inventor, existe uma guia chamada de `Blocks` , e nela temos uma seção chamada de `Built-in` , conforme demonstra a figura adiante. Dentro dela, existem algumas opções que vamos conhecer assim que surgir a necessidade. Mas para este momento, clique na opção `Variables`

para abrir uma outra guia com todas as opções das variáveis.

O QUE SÃO AS VARIÁVEIS?

São espaços reservados na memória do computador para guardar informações que serão usadas em outros momentos durante a execução do aplicativo.

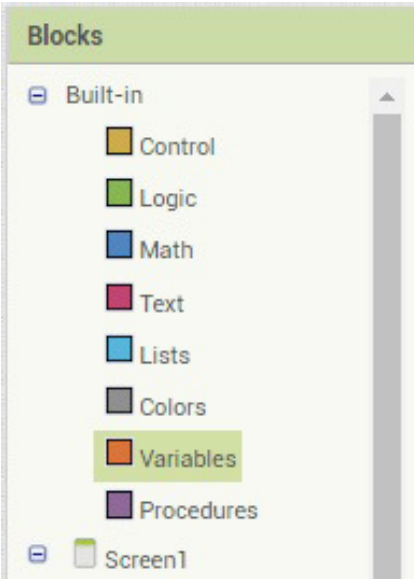

Figura 3.23: Guia Built-in

Figura 3.24: Opção Variables selecionada

Como necessitamos criar duas variáveis, clique na opção `Initialize global name to` para ela ser inserida na área `Viewer`, e repita o processo para inserir um segundo bloco de variável. A figura a seguir exibe os dois blocos das variáveis:

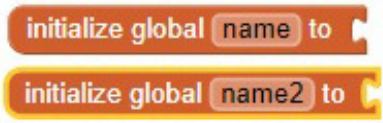

Figura 3.25: Variáveis inseridas

Sempre que criar uma variável, torna-se necessário definir também o tipo de informação que ela vai armazenar. No exercício atual, vamos armazenar os **números** que o usuário digitará nas `TextBoxs`, por isso a nossa variável será do tipo **numérica**. Para isso, deveremos inserir um bloco com o valor `0` (zero) e encaixar

na sequência `Initialize global name to` .

> Variável do tipo numérica apenas armazena números. Encaixar um bloco com o valor `0` (zero) na criação da variável torna-a do tipo numérica.

Vá até a seção `Built-in` e verifique que existe uma opção chamada `Math` . Nela estão todas as opções matemáticas disponíveis no App Inventor:

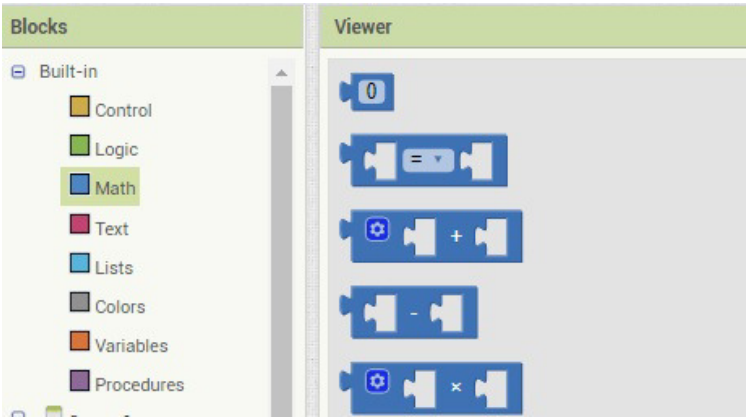

Figura 3.26: A guia Math

Como queremos criar variáveis numéricas para armazenar os valores digitados pelo usuário, necessitamos inserir um bloco que indique o **número zero**. Clique e arraste-o para encaixar na variável. Veja na próxima imagem como ela ficará. Repita esse procedimento para a criação de uma segunda variável numérica.

Figura 3.27: Variável numérica

Toda variável necessita de um nome próprio que nos ajudará na sua identificação durante o restante do desenvolvimento do aplicativo. Vamos trocar os nomes para `numero1` e `numero2`, pois são bem melhores do que os nomes de identificação sugeridos (`name` e `name2`).

Para renomear uma variável, basta clicar sobre o nome que consta no bloco da variável `name` e digitar o novo (`numero1`). Repita esse procedimento para a segunda variável renomeando-a para `numero2`. A figura a seguir exibe as suas variáveis numéricas finalizadas.

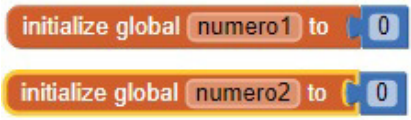

Figura 3.28: Variáveis numéricas finalizadas

3.3 PROGRAMANDO UM BOTÃO

Para dar funcionalidade para cada um dos botões que inserimos na tela de designer, necessitamos utilizar um bloco que represente a ação que o usuário realizou. Essa ação é o clique que o botão receberá. Existe um bloco que representa essa ação, o `When Btn_Soma.click`.

Para localizá-lo, vá até a guia `Blocks` na lateral esquerda da tela e localize o nome que demos ao botão que realiza a soma, ou

seja, o `Btn_Soma` . Ao selecioná-lo, surgirá um complemento de tela ao lado direito com todas as possibilidades de ações que o botão poderá sofrer. A figura a seguir exibe a tela após a seleção do bloco `Btn_Soma` .

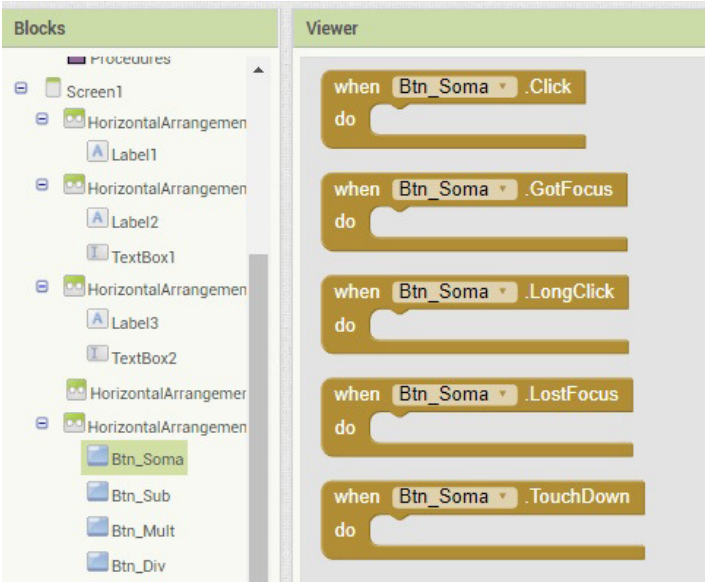

Figura 3.29: Opções do Btn_Soma

Como queremos programar as funções de soma, teremos de inserir um bloco que represente **quando o Btn_Soma for clicado**. Localize o bloco `When Btn_Soma.click` e arraste-o para a área `Viewer` . A próxima imagem exibe o `Btn_Soma` nessa área.

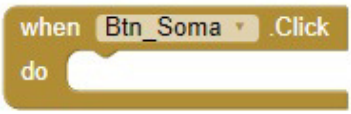

Figura 3.30: Btn_Soma

Agora, as variáveis que foram criadas e definidas como `numero1` e `numero2` deverão receber os valores digitados e contidos nas `TextBoxs` de nomes `Txt_N1` e `Txt_N2` .

Para uma variável receber um valor vindo de outro bloco, é preciso usar o comando `set to` . Acesse a área `Built-in` e, na seção `Variables` , clique sobre o bloco `set to` e coloque-o dentro do bloco `When Btn_Soma.click` , pois somente quando o clique no botão acontecer é que a variável receberá um valor. A figura a seguir exibe o bloco da variável dentro do botão.

Figura 3.31: Btn_Soma

Note que o local onde fica visível o nome da variável está vazio. Precisamos indicar o nome da variável que receberá o primeiro valor da `TextBox` . Para isso, clique no espaço logo após o comando `set` e selecione o nome `global numero1` :

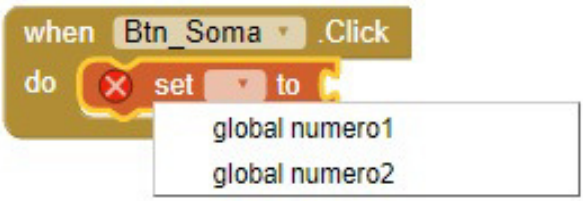

Figura 3.32: Selecionando o nome da variável

Após a seleção, a variável deverá ficar assim:

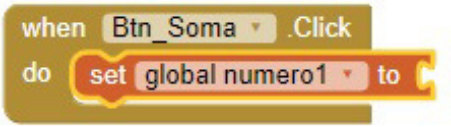

Figura 3.33: Variável selecionada

A variável `numero1` deverá receber o valor que foi digitado e está na `Txt_N1` . Deveremos inserir um bloco `Txt_N1.Text` , pois é nele que realmente se encontra o valor digitado. Localize na área de `Blocks` o componente `Txt_N1` . Clique sobre ele para expandir a tela com todos os seus possíveis blocos. A figura a seguir demonstra a expansão da janela.

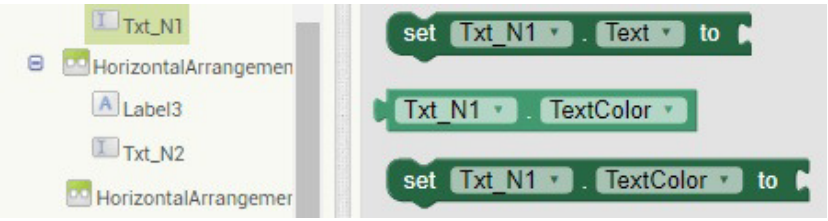

Figura 3.34: Opções de blocos da Txt_N1

Localize a janela em que surgiu o bloco `Txt_N1.Text` . Clique

e arraste-o para dentro do bloco da variável `set global numero1 to` . Veja esse procedimento na próxima figura.

Figura 3.35: Opções de blocos da Txt_N1

Repita o procedimento realizado com a variável `numero1` para a variável `numero2` . A figura exibe o bloco com as duas variáveis recebendo os valores:

Figura 3.36: Variáveis recebendo os valores digitados

Já temos os valores informados atribuídos às variáveis. Agora precisamos exibir o resultado da soma das variáveis em uma `Label` . O bloco de uma `Label` para exibição de valores é o `set Lbl.Resultado.Text to` . Localize na seção de blocos o componente `Lbl_Resultado` para expandir os possíveis blocos ao lado direito. A figura a seguir exibe os blocos do `Lbl_Resultado` .

Figura 3.37: Opções de blocos do Lbl_Resultado

Selecione e arraste o bloco `set Lbl.Resultado.Text to` para dentro do botão `Btn_Soma` :

Figura 3.38: Opções de blocos do Lbl_Resultado

Deveremos realizar a soma antes da exibição. Como já vimos anteriormente, na guia `Built-in` existe a seção `Math` ; selecione-a para expandir e ver as possibilidades de operações matemáticas existentes. Lá você encontrará o bloco que realiza a adição:

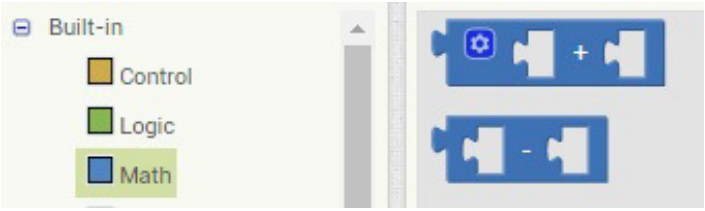

Figura 3.39: Opções de blocos do Lbl_Resultado

Selecione o bloco de adição e encaixe-o logo após a `Label` que exibirá o resultado, pois o resultado da soma das duas variáveis será exibido na `Lbl_Resultado` . A figura a seguir exibe o bloco de adição encaixado no bloco de exibição do resultado.

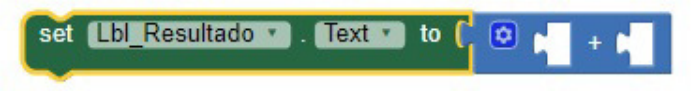

Figura 3.40: Bloco de adição posicionado

Note que, dentro do bloco de adição, existem dois espaços em branco para encaixar os blocos das variáveis que serão somadas. Vamos selecionar uma variável para posicionar no primeiro espaço do bloco de adição.

Localize na área `Viewer` o bloco da criação da variável `numero1` . Para indicar a variável que desejamos, posicione o ponteiro do mouse sobre seu nome para ver as possibilidades que surgirão na tela. A figura a seguir exibe essas possibilidades.

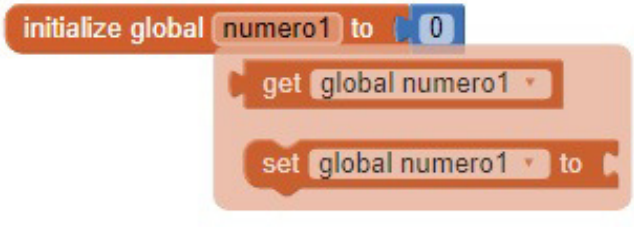

Figura 3.41: Possibilidades das variáveis

- `get global numero1` — Indica que usaremos o valor que está armazenado na variável.
- `set global numero1 to` — Indica que estaremos

armazenando um novo valor na variável.

Selecione o bloco `get global numero1` e encaixe no primeiro espaço vazio do bloco de adição:

Figura 3.42: Posicionando a variável para somar

Repita o procedimento realizado com a variável `numero1` para a variável `numero2`, e encaixe-a no segundo espaço do bloco de adição. Ao término, o bloco de exibição da soma deverá estar igual ao exibido na figura:

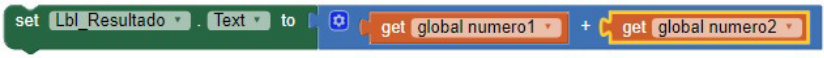

Figura 3.43: Bloco da soma

A próxima figura exibe o bloco do botão `Btn_Soma` finalizado.

when Btn_Soma .Click
do
set global numero1 to Txt_N1 . Text
set global numero2 to Txt_N2 . Text
set Lbl_Resultado . Text to
get global numero1 +
get global numero2

Figura 3.44: Btn_Soma finalizado

Desafio

Para os botões de subtração, multiplicação e divisão, o procedimento é basicamente idêntico ao realizado com o botão de soma. Será modificado apenas o bloco da guia `Math` para realizar a operação desejada. Seguindo as orientações deste capítulo, deixo

aqui um desafio para o leitor: complete os demais blocos com as operações faltantes.

3.4 TESTANDO O APLICATIVO

Após a criação de todos os botões, chegou a hora de testar seu app. Não se esqueça de primeiro clicar no aplicativo **aiStarter** para habilitá-lo. Como vimos no capítulo anterior, para emular seu app, clique no menu suspenso `Connect` do App Inventor e selecione a opção `Emulator` , então é só aguardar o carregamento do seu aplicativo para testá-lo.

Digite alguns valores nas caixas de texto e clique no botão desejado para fazer a operação matemática. No nosso exemplo, foram digitados os valores 5 e 8, e depois clicado no botão somar, apresentando na `Label` `Lbl_Resultado` a soma 13. A figura a seguir exibe a tela emulada com os valores digitados.

Figura 3.45: Teste do aplicativo

3.5 RESUMINDO

Neste capítulo, conhecemos e aprendemos sobre os componentes `Label` e `Button` e suas diversas propriedades de configuração. Nos blocos, aprendemos a declarar as variáveis, utilizar os blocos da guia `Math` e programar os botões para a exibição do resultado da operação matemática selecionada.

No próximo capítulo, vamos criar um aplicativo usando comandos de decisão, com duas telas que nos ajudarão a decidir qual combustível é mais vantajoso para abastecer o seu veículo:

etanol ou gasolina?

Capítulo 4

ETANOL OU GASOLINA?

Se você possui um veículo *flex*, deve ter reparado na diferença flutuante entre os valores dos combustíveis etanol e gasolina, e deve ter se perguntado com qual combustível deveria abastecer. Sabendo que, com um litro de etanol, você roda uma quilometragem menor do que com um litro de gasolina, qual seria o mais vantajoso? Neste capítulo, construiremos um aplicativo para lhe ajudar a decidir entre abastecer com etanol ou gasolina.

Nesse aplicativo, teremos uma tela de interação para a digitação dos valores dos dois combustíveis e um botão para realizarmos os cálculos e exibirmos qual é a melhor opção de abastecimento. Além disso, teremos também um botão para limpar os dados digitados e um para acessar uma segunda tela, para que o usuário saiba como é feito o cálculo para a escolha do combustível.

4.1 TELA DO APLICATIVO

Veja na figura a seguir como será a tela final do nosso aplicativo:

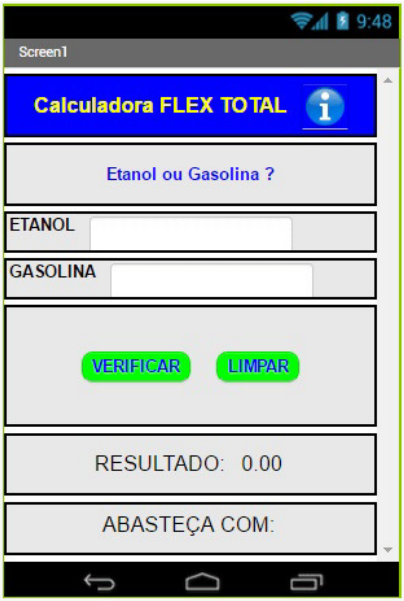

Figura 4.1: Tela final do aplicativo Calculadora Flex Total

Para começar, vamos criar um novo projeto. Para isso, clique no menu `Projects`, selecione a opção `Start New Project`, e dê o nome de `Etanol_ou_Gasolina`.

Vamos começar a desenvolver nossa tela inicial. Como vimos na figura, necessitamos de três **botões**: um para mostrar a fórmula do cálculo; outro para realizar os cálculos e exibir o resultado desejado; e um para limpar os dados da tela para realizarmos um novo cálculo.

Necessitamos também de duas **caixas de texto** para a digitação dos valores do etanol e da gasolina, que serão utilizadas para a realização dos cálculos. Por último, vamos precisar também de algumas `Labels` para exibir os valores resultantes das contas e da

decisão realizada pelo aplicativo.

Veja na tabela adiante os componentes que devemos inserir, a guia em que eles se encontram e as propriedades que deverão ser alteradas para conseguirmos construir a tela do aplicativo.

Observe que, na coluna **Componente**, estão todos os itens que devemos arrastar para a tela do aplicativo. Já a coluna **Pallete** nos indica o local onde encontramos os componentes, que estão agrupados por categorias de funcionalidades. Por exemplo, para acessar a `Pallete Layout` e encontrar um componente `HorizontalArrangement`, basta clicar sobre o nome dessa `Pallete` e todos os componentes responsáveis pelo ajuste do layout estarão visíveis.

A coluna **Propriedade** indica qual propriedade deverá ser alterada, pois nem sempre precisamos alterar todas as propriedades de um componente. Já a coluna **Alterar valor** indicará qual o novo valor que cada uma deverá receber para chegarmos à tela final do nosso aplicativo.

Antes de inserir os componentes, perceba que já os utilizamos e explicamos no capítulo anterior. Com isso, o leitor já deverá estar mais familiarizado com o ambiente de desenvolvimento. Mas vamos começar a desenvolver o layout de nosso app.

Com a sua `Screen1` selecionada, altere a propriedade `AppName` para **Etanol ou Gasolina?**, pois este será o nome exibido no aplicativo quando ele estiver instalado em seu dispositivo. Criaremos a nossa própria área de título do aplicativo, então para o App Inventor não exibir o título padrão, desmarque a opção na propriedade `TitleVisible`.

Com essas propriedades já alteradas, poderemos começar a desenvolver a barra de título de nosso app. Ela terá uma `Label` com o nome do aplicativo e um `Button` que abrirá uma segunda tela, com as informações sobre como é feito esse cálculo para a decisão entre os combustíveis. Então, para exibir dois ou mais componentes na mesma linha, o App Inventor necessita que insiramos um controle `HorizontalArrangement`. É ele quem faz o **arranjo horizontal** dos componentes na linha.

Após inserir um componente `HorizontalArrangement`, precisaremos realizar alguns ajustes em suas propriedades. Veja na tabela a seguir as propriedades que deverão ser alteradas e seus novos valores.

Componente	Pallete	Propriedade	Alterar valor
HorizontalArrangement	Layout	AlignHorizontal	Center
		AlignVertical	Center
		BackgroundColor	Blue
		Height	50 pixels
		Width	Fill Parent

A propriedade `AlignHorizontal` realiza o alinhamento dos componentes que ficarão em seu interior. Aqui selecionamos o alinhamento para que todos os componentes fiquem alinhados no centro da tela, por isso foi marcada a opção `Center`. Agora para um alinhamento referente ao seu espaço vertical, dentro do `HorizontalArrangement`, a propriedade `AlignVertical` também é marcada como `Center`.

A propriedade `BackgroundColor`, que indicará a cor de fundo da barra de título desenvolvida, deverá ficar com uma cor

diferente da utilizada no fundo, então escolhemos a cor `Blue` . Necessitamos também alterar os tamanhos da altura e largura que o `HorizontalArrangement` ocupará dentro da `Screen1` . Para alterar a altura da barra de título, usaremos na propriedade `Height` o valor de `50 pixels` e, para o seu comprimento, na propriedade `Width` , selecione a opção `Fill Parent` que a deixará ocupando todo o espaço horizontal do seu dispositivo.

Preparado o espaço para o título, devemos inserir em seu interior uma `Label` para exibir o título do aplicativo e um botão que, ao ser pressionado, exibirá uma outra tela com as informações do aplicativo. A tabela seguinte indica a ordem em que devemos inserir os componentes e as propriedades que deverão ser alteradas.

Componente	Pallete	Propriedade	Alterar valor
Label	User Interface	FontBold	Marcar
		FontSize	20
		Text	Calculadora FLEX TOTAL
		TextColor	Yellow
Button	User Interface	Height	40 pixels
		Width	40 pixels
		image	info.png
		Text	Apagar o texto
		Renomear	Btn_Informacoes

Para a `Label` , selecionamos a propriedade `FontBold` , pois ela deixará o título com o efeito de **negrito**. Para aumentar o tamanho do texto, modificamos a propriedade `FontSize` para o

valor de 20. A propriedade que exibe o texto que desejamos exibir é a `Text`, onde foi digitado o título **Calculadora FLEX TOTAL**. Em `TextColor`, alteramos a cor do texto para `Yellow`.

Já para o componente `Button`, foi definida a altura do botão para `40 pixels` através da propriedade `Height`. Para a largura do botão, também deixamos a propriedade `Width` para `40 pixels`. Inserimos a imagem `info.png` conforme demonstrado no capítulo anterior. Apagamos também a propriedade `Text` do botão para ele possuir somente a imagem. Não se esqueça de que alteramos o nome do componente para `Btn_Informacoes`.

Veja como ficou a barra de título de seu aplicativo:

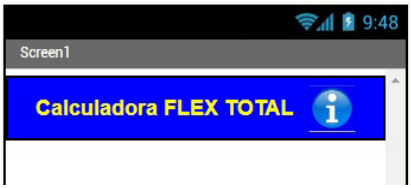

Figura 4.2: Barra de título do aplicativo

Continuando o desenvolvimento do layout, vamos agora exibir uma mensagem para o usuário: **Etanol ou Gasolina?**. Devemos primeiramente inserir um componente `HorizontalArrangement` logo abaixo da barra de título e, dentro dele, uma `Label` para realizar a exibição da mensagem. Após inserir os objetos na sua `Screen1`, realize as configurações conforme demonstrado a seguir.

Componente	Pallete	Propriedade	Alterar valor
HorizontalArrangement	Layout	AlignHorizontal	Center
		AlignVertical	Center

		Height	50 pixels
		Width	Fill Parent
Label	User Interface	FontBold	Marcar
		FontSize	16
		Text	Etanol ou Gasolina?
		TextColor	Blue

A figura a seguir exibe o atual estágio da tela do app:

Figura 4.3: Mensagem: Etanol ou Gasolina?

Devemos agora preparar a digitação do valor do **Etanol**. Precisaremos de uma `Label` para informar a legenda do que devemos digitar e de um componente `TextBox` que será responsável por receber o valor digitado pelo usuário. Não podemos nos esquecer que tanto a `Label` como a `Textbox` deverão ser inseridas dentro de um `HorizontalArrangement` . Comece inserindo o componente `HorizontalArrangement` da guia `Layout` logo abaixo do `HorizontalArrangement` que contém a mensagem *Etanol ou Gasolina?*, e realize as configurações que estão na tabela a seguir.

Componente	Pallete	Propriedade	Alterar valor

HorizontalArrangement	Layout	Width	Fill Parent
		AlignVertical	Center
Label	User Interface	FontBold	Marcar
		FontSize	16
		Text	ETANOL
TextBox	User Interface	Hint	Preço do Etanol
		NumbersOnly	Marcar
		TextAlignment	Right
		Renomear	Txt_Etanol

Vamos comentar apenas as propriedades que ainda não utilizamos, pois as demais continuam a realizar as mesmas funções já vistas anteriormente. No componente `HorizontalArrangement`, alteramos a propriedade `AlignVertical` que centralizará verticalmente todos os demais componentes que estiverem em seu interior. A propriedade `NumbersOnly` do componente `TextBox`, quando marcada, ativará apenas o teclado numérico do telefone ou dispositivo em que o aplicativo estiver instalado, pois o teclado alfanumérico torna-se desnecessário nesse momento, já que desejamos digitar apenas números. Alteramos também a propriedade `TextAlignment` para exibir as informações alinhadas ao lado direito do componente `TextBox`. A figura seguinte exibe a tela com o espaço para a digitação do valor do etanol.

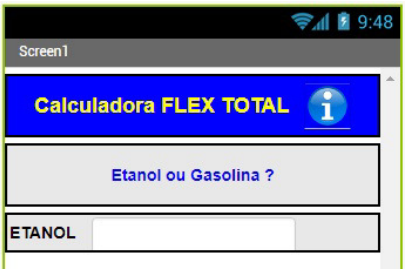

Figura 4.4: Digitação do valor do Etanol

Seguindo essa mesma ideia, realize a inserção e configuração dos componentes para a digitação do preço da **Gasolina**. Veja na tabela os componentes que deverão ser inseridos e as propriedades com os valores para alterar:

Componente	Pallete	Propriedade	Alterar valor
HorizontalArrangement	Layout	Width	Fill Parent
		AlignVertical	Center
Label	User Interface	FontBold	Marcar
		FontSize	16
		Text	GASOLINA
TextBox	User Interface	Hint	Preço da Gasolina
		NumbersOnly	Marcar
		TextAlignment	Right
		Renomear	Txt_Gasolina

A figura a seguir exibe a tela do app após inserirmos os componentes descritos anteriormente.

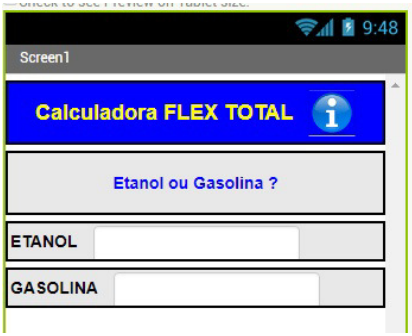

Figura 4.5: Tela com os botões de controle

Necessitamos agora inserir mais um `HorizontalArrangement` abaixo do espaço reservado para a digitação do preço da gasolina, pois ele vai conter dois botões de controle em seu interior. O primeiro realizará o cálculo para informar qual combustível é mais vantajoso para se abastecer, e o segundo realizará a limpeza das informações. Entre eles, vamos inserir uma `Label` que não terá informação nenhuma, com a finalidade de inserir um espaçamento estético entre os botões de verificar e limpar.

A tabela a seguir exibe a ordem em que deveremos inserir os componentes descritos anteriormente e as propriedades que precisamos alterar juntamente com os novos valores.

Componente	Pallete	Propriedade	Alterar valor
HorizontalArrangement	Layout	AlignVertical	Center
		AlignHorizontal	Center
		Height	100 pixels
		Width	Fill Parent

Componente	Pallete	Propriedade	Alterar valor
Button	User Interface	BackgroundColor	Green
		FontSize	16
		FontBold	Marcar
		Shape	Rounded
		Text	VERIFICAR
		TextAlignment	Center
		TextColor	Blue
		Renomear	Btn_Verifica
Label	User Interface	Text	Inserir 5 espaços
Button	User Interface	BackgroundColor	Green
		FontSize	16
		FontBold	Marcar
		Shape	Rounded
		Text	LIMPAR
		TextAlignment	Center
		TextColor	Blue
		Renomear	Btn_Limpar

A única propriedade que ainda não utilizamos é a `Shape`, que está presente na configuração dos componentes `Buttons`. A seleção da opção `Rounded` é responsável por exibir um contorno arredondado do botão, trocando o modelo padrão retangular.

Também é preciso destacar que deixamos a propriedade `Text` da `Label` com cinco espaços (pressione 5 vezes a barra de espaço de seu teclado). Utilizamos este recurso com a finalidade de inserir um espaçamento estético entre os botões de verificar e limpar.

A figura a seguir exibe a tela após inserirmos os botões e configurarmos as suas propriedades.

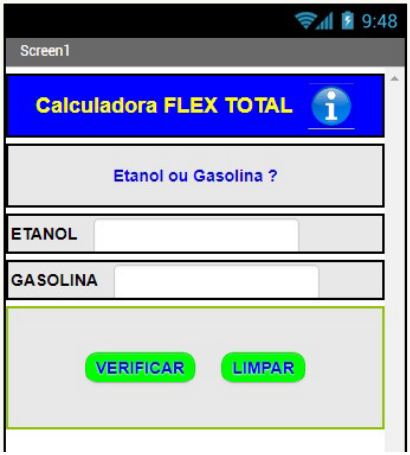

Figura 4.6: Tela com os botões de controle

Quando o usuário pressionar o botão `Btn_Verifica`, precisaremos de uma `Label` para exibir o resultado da conta que será demonstrado mais adiante. Teremos um `HorizontalArrangement` e duas `Labels` em seu interior: uma para exibir a palavra **Resultado** e outra para exibir o resultado do cálculo que efetuaremos mais adiante, neste capítulo.

A tabela seguinte indica a ordem em que os componentes deverão ser inseridos e as suas propriedades com os novos valores para alteração.

Componente	Pallete	Propriedade	Alterar valor
HorizontalArrangement	Layout	AlignVertical	Center
		AlignHorizontal	Center
		Height	50 pixels

		Width	Fill Parent
Label	User Interface	FontSize	20
		Text	RESULTADO
Label	User Interface	FontSize	20
		Text	0.00

Note que não há nenhuma nova propriedade, e isso permitirá ao leitor realizar esta tarefa como um pequeno teste de seu aprendizado. A figura a seguir exibe a tela após inserirmos a `Label` de resultado.

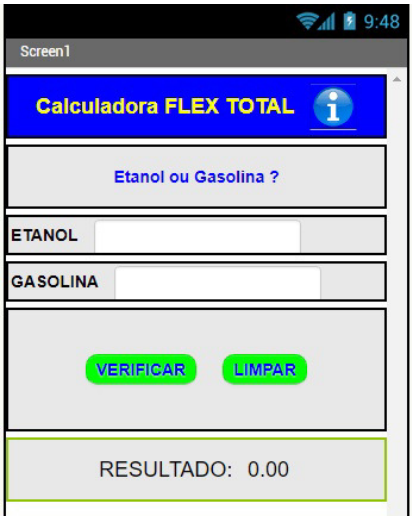

Figura 4.7: Tela com Labels de resultado

Agora, necessitamos informar qual combustível será mais vantajoso após realizarmos os cálculos. Para tanto, vamos inserir novamente mais duas `Labels` : a primeira para exibir a mensagem **Abasteça com** e a segunda para indicar uma das opções, **Etanol** ou

Gasolina. Porém, para demonstrar um novo componente, em vez do `HorizontalArrangement`, veremos o componente `VerticalArrangement`. Ele tem a função de organizar dois ou mais componentes verticalmente em sua tela.

Insira o `VerticalArrangement` abaixo do último `HorizontalArrangement` e, dentro dele, insira as duas `Labels`. A tabela a seguir indica as propriedades com os novos valores para alteração.

Componente	Pallete	Propriedade	Alterar valor
VerticalArrangement	Layout	AlignVertical	Center
		AlignHorizontal	Center
		Height	Fill Parent
		Width	Fill Parent
Label	User Interface	FontSize	20
		Text	ABASTEÇA COM
Label	User Interface	FontBold	Marcar
		FontSize	50
		TextColor	Red
		Text	--
		Renomear	Lbl_Resultado

Figura 4.8: Tela principal finalizada

Para o aplicativo emitir uma mensagem ao usuário, no caso dele se esquecer de digitar algum valor, vamos acrescentar um componente que realiza essa tarefa, o `Notifier`. Você encontra-o na guia `User Interface` e não precisa alterar nenhuma propriedade. Esse componente é chamado de componente não visível, pois ele não será exibido na sua `Screen1` durante o desenvolvimento do layout, ficando localizado na seção de `Non-visible components`.

Antes de programar os blocos — que são representações gráficas de comandos que executam determinadas tarefas no aplicativo —, veja a seguir a fórmula que usaremos para calcular qual combustível é mais vantajoso para abastecer:

Fórmula

Resultado = Etanol / Gasolina

Se o resultado obtido for menor ou igual a 0.70, isso indicará que abastecer com etanol é mais vantajoso; caso contrário, o app deverá mostrar que a gasolina será mais indicada economicamente.

4.2 INSERINDO OS BLOCOS DE CONTROLE

Agora vamos para a área de programação. Para isso, basta clicar no botão `Blocks` no canto superior direito de sua tela:

Figura 4.9: Botão para acionar a área de blocos

Precisamos definir três variáveis numéricas para receber e armazenar os valores digitados do preço do etanol e da gasolina, além do resultado do cálculo obtido através da fórmula apresentada.

Como vimos no capítulo anterior, acesse a área `Blocks` e, na guia `Built-in`, selecione três `initialize global name to` na opção `Variables`, conforme demonstra a figura a seguir. Esses blocos têm por finalidade declarar as variáveis que usaremos.

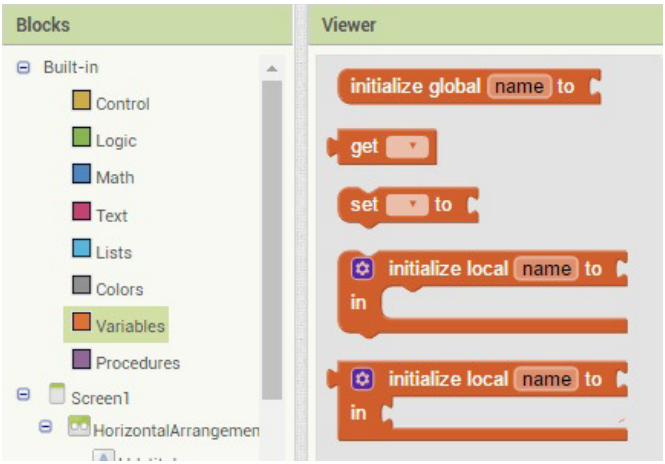

Figura 4.10: Inserindo variáveis

Vamos trocar os nomes das variáveis criadas em cada um de nossos blocos. Clique onde você lê a palavra `name` e altere para **Etanol**, e faça esse procedimento para os outros dois blocos, digitando os nomes **Gasolina** e **Resultado**. Veja os três blocos das variáveis sem os nomes alterados na próxima figura.

Figura 4.11: Blocos das variáveis

Encaixe o valor zero (0) em cada uma das variáveis criadas, para indicar que elas serão numéricas:

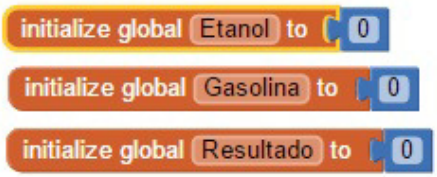

Figura 4.12: Variáveis do app

Vamos agora trabalhar com o botão que realiza os cálculos e a verificação de qual combustível é mais vantajoso. Precisamos inserir um bloco que represente o clique no `Btn_Verifica`, para que os cálculos sejam executados. Devemos primeiramente selecionar na guia de `Blocks` o componente `Btn_Verifica`.

Note que, ao clicar sobre o `Btn_Verifica`, um leque de ações que esse botão pode executar se abrirá ao lado direito. Selecione então o bloco `when Btn_Verifica.Click`, conforme demonstra a figura a seguir. Esse bloco de ação — ou como chamamos na programação, esse **evento** — nos indica que, quando o botão de verificar for clicado, serão executados os demais blocos que estão em seu interior.

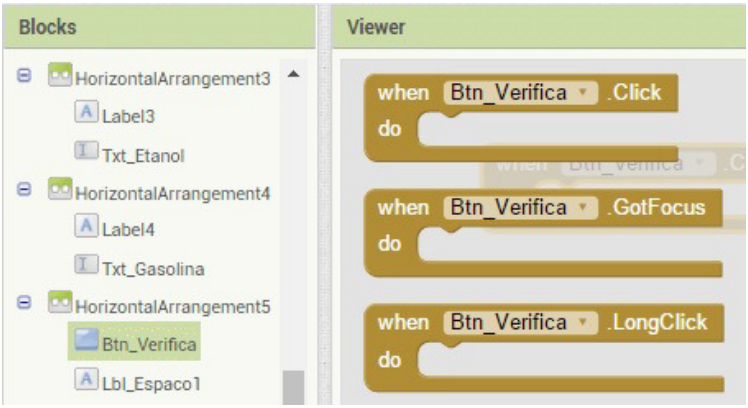

Figura 4.13: Selecionando o bloco Btn_Verifica.Click

Para que o aplicativo identifique o clique do botão `Btn_Verifica`, devemos selecioná-lo e arrastá-lo para a área de programação, ou seja, a área `Viewer` do App Inventor, como demonstrado na figura:

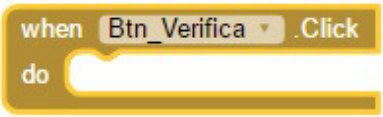

Figura 4.14: Bloco Btn_Verifica

4.3 TOMANDO A DECISÃO

Para o App Inventor poder decidir qual combustível é mais vantajoso, precisamos de um bloco de controle que faça essa verificação. Esse processo é realizado pelo bloco `if`. Junto a este bloco, sempre devemos anexar uma ou mais comparações. No caso de a decisão ser verdadeira, será executada a opção ou opções que vierem encaixadas na frente do comando `then`. No caso de a condição ser falsa, nenhum bloco será executado e a programação continuará apenas com os blocos que estiverem abaixo do bloco `if`.

O bloco `if`, ou estrutura de decisão, possui duas maneiras de trabalhar: o método simples, que é o que acabamos de demonstrar e trata apenas de realizar as tarefas quando a verificação for verdadeira; e o método composto que, além de tratar a condição verdadeira, também executa alguma ação se a comparação do `if` for falsa. Para isso, devemos inserir uma configuração no bloco `if`, ou seja, incluir a opção `else`.

Dentro do `Btn_Verifica`, arraste um bloco de controle `if`. Para isso, selecione na guia `Built-in` a opção `Control` que, ao lado direito, aparecerá todos os controles disponíveis, então selecione e arraste a opção de decisão `if` para dentro do bloco do `Btn_Verifica`. Veja na figura a seguir como inserir o bloco `if`.

Figura 4.15: Bloco if

Esse bloco terá a finalidade de verificar se o usuário digitou os valores do etanol e da gasolina. Caso o usuário não digitar algum dos dados, o aplicativo emitirá uma informação para que ele digite todos os valores; caso contrário, os cálculos serão realizados. Para isso, precisamos configurar o bloco `if` para aceitar a opção contrária (`else`).

Veja com inserir o `else` em um bloco `if`:

1. Primeiro, clique no ícone de cor azul ao lado esquerdo do comando `if`:

Figura 4.16: Configurando o else

2. Clique no comando `else`, segure e arraste-o para o lado

esquerdo dentro do `if` . A figura a seguir exibe o comando `else` , após clicar no ícone de configuração do bloco `if` .

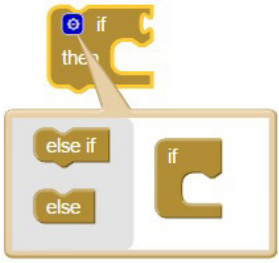

Figura 4.17: Inserindo o bloco else

3. Veja como deverá ficar seu bloco `if/else` finalizado:

Figura 4.18: Bloco if/else finalizado

O bloco `Btn_Verifica` deverá ficar com o `if` inserido desta forma:

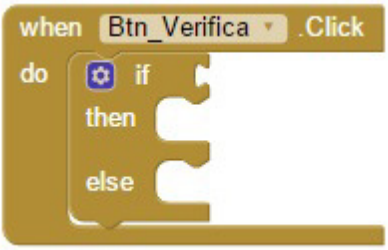

Figura 4.19: Btn_Verifica com o bloco if

Com esse bloco de decisão já posicionado na área `Viewer`, precisamos de um bloco para decidir se a primeira **ou** a segunda opção está sem receber um valor. Para isso, vamos inserir um bloco `or`, da guia `Logic` da seção `Built-in`, e encaixá-lo no bloco `if`. Veja o resultado na figura a seguir:

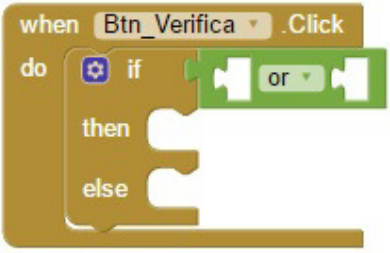

Figura 4.20: Bloco lógico or encaixado no if

Já temos o bloco `if` que realiza a decisão, porém falta ainda inserir seus critérios. Como queremos verificar se não foi digitado algum valor nos campos do etanol ou da gasolina, precisamos inserir um bloco `is empty` da guia `Text`, ou seja, verificar se os campos dos valores estão vazios. Posicione-o para dentro do bloco lógico `or`, conforme vemos a seguir.

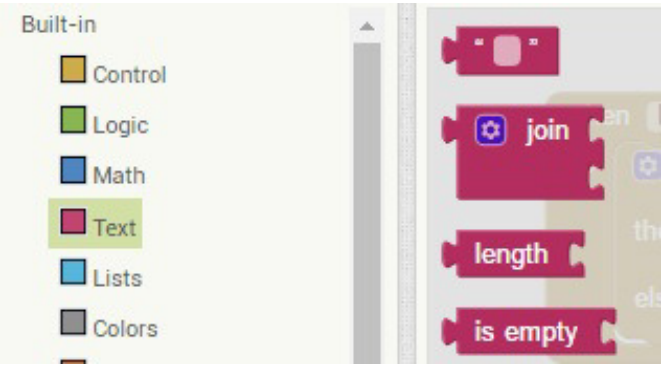

Figura 4.21: Selecionando o bloco is empty

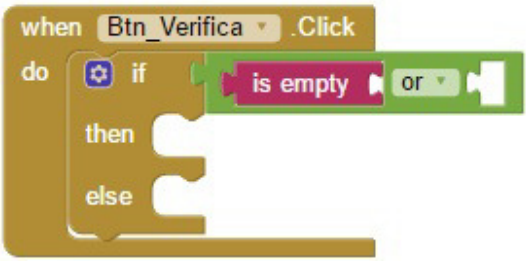

Figura 4.22: Bloco is empty encaixado no if

Agora que já inserimos o bloco `is empty` , devemos verificar qual componente não poderá ficar vazio. Então selecione o componente `Txt_Etanol.Text` e encaixe-o logo após o bloco `is empty` . Com a junção desses blocos, conseguimos verificar se não foi digitado um valor para o campo etanol. Não podemos esquecer de realizar a mesma verificação para o valor da gasolina, para isso, repita o procedimento inserindo o bloco do componente `Txt_Gasolina.Text` após o `is empty` . A figura a seguir demonstra a linha do `if` finalizada.

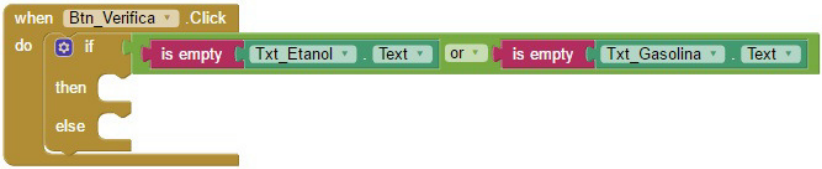

Figura 4.23: Verificando se os valores não foram digitados

Após essa verificação, e se ela for verdadeira, devemos exibir uma notificação ao usuário para que digite as informações necessárias. Existe um componente que realiza essa tarefa, o `notifier`. Note que já inserimos esse componente no design do projeto. Então, vamos adicionar um bloco `call notifier.ShowAlert` para encaixar na frente da opção `then` do `if`, pois este é o local que devemos programar quando a decisão testada por esse bloco for considerada verdadeira.

Na sequência do `Notifier1`, insira um bloco de texto e digite a informação que desejamos exibir para o usuário: `Digite todos os valores!`.

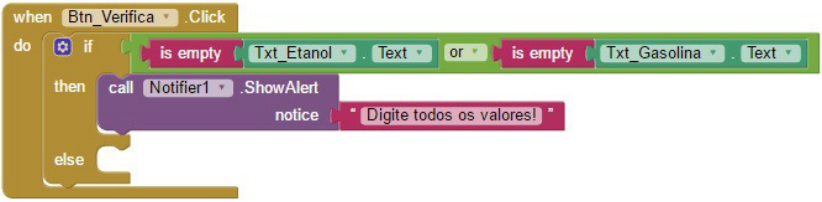
when Btn_Verifica .Click
do if is empty Txt_Etanol . Text or is empty Txt_Gasolina . Text
then call Notifier1 .ShowAlert
notice " Digite todos os valores! "
else

Figura 4.24: Exibindo uma notificação ao usuário

Caso o usuário tenha digitado os dois valores, as variáveis etanol e gasolina deverão recebê-los. Para transferir os valores digitados em tela para as variáveis, devemos realizar uma atribuição de valores, isto é, fazer cada uma delas receber o seu

valor específico.

Então, devemos primeiramente inserir um bloco para receber o valor. Para isso, posicione o ponteiro do mouse sobre a variável `Etanol` para aparecer a opção `set global Etanol to` , já que a opção `set global` nos indica que a variável receberá um valor. Depois, arraste-o para encaixar na frente da opção `else` do bloco `if` , pois só poderemos receber os valores se não estiverem vazios (`else`). A figura a seguir mostra como exibir as opções da variável `Etanol` .

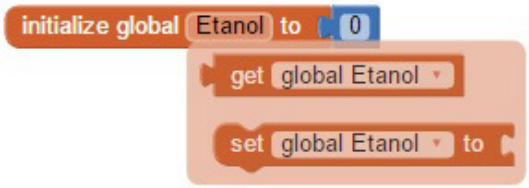

Figura 4.25: Exibindo as opções da variável Etanol

Encaixe-o na variável em que você inseriu um componente `Txt_Etanol.text` , pois é nele que se encontra o valor digitado pelo usuário, e é nesse exato momento que ocorre a atribuição de valor da informação digitada pelo usuário para a variável do aplicativo. Você precisará realizar o mesmo procedimento com a variável `Gasolina` , encaixando nela o bloco `Txt_Gasolina` . Veja na próxima figura o resultado parcial dos blocos.

when Btn_Verifica .Click
do if is empty Txt_Etanol . Text or is empty Txt_Gasolina . Text
then call Notifier1 .ShowAlert
notice " Digite todos os valores! "
else set global Etanol to Txt_Etanol . Text
set global Gasolina to Txt_Gasolina . Text

Figura 4.26: Movendo os dados para as variáveis

Já temos as variáveis com os valores digitados. Agora precisamos realizar a conta com a fórmula especificada, em que a variável do resultado receberá a divisão dos valores das variáveis `Etanol` pela `Gasolina`. Para podermos usar o valor que está atribuído a uma variável, devemos utilizar o comando `get global`.

> **Resumindo**
>
> O comando `set` indica que a variável está recebendo um valor. Já o comando `get` indica que queremos utilizar o valor atribuído à variável.

Logo após os blocos de recebimento dos valores digitados, colocaremos um bloco da variável para receber o resultado da conta. Encaixe um bloco da variável `set global resultado to`.

Precisamos de um bloco que realize uma operação matemática de divisão. Note que, na guia `Built-in`, temos uma opção chamada `Math`. Nela encontramos todas as operações matemáticas disponíveis no App Inventor. Localize nessa opção

um bloco de divisão para encaixar na frente da variável resultado. Dentro dos espaços no bloco de divisão, insira um bloco de variável `get global Etanol` e um da `get global Gasolina`. Estes dois últimos você encontra posicionando novamente o ponteiro do mouse sobre a definição da variável, conforme já demonstrado. A próxima figura exibe o bloco que calcula a divisão dos preços.

when Btn_Verifica .Click
do if is empty Txt_Etanol . Text or is empty Txt_Gasolina . Text
then call Notifier1 .ShowAlert
notice " Digite todos os valores! "
else set global Etanol to Txt_Etanol . Text
set global Gasolina to Txt_Gasolina . Text
set global Resultado to get global Etanol / get global Gasolina

Figura 4.27: Cálculo da divisão do preço do etanol pelo da gasolina

Já calculamos a divisão dos preços do etanol pela gasolina, agora precisamos exibir essa informação no aplicativo para que o usuário possa vê-la. Anteriormente, inserimos um componente `Lbl_Porcento.Text` para isso. Vamos inserir o bloco do componente `set Lbl_Porcento.Text to` da guia `Blocks` para receber o resultado do cálculo.

Como o resultado da divisão pode conter várias casas decimais após a vírgula, seria muito interessante, por questão apenas de estética, que formatássemos a exibição para mostrar apenas dois números depois dela. Para isso, devemos inserir um bloco que vai configurar a exibição do resultado com duas casas decimais.

Selecione na `Built-in` da guia `Math` a opção `format as`

`decimal number` , responsável pela formatação que queremos. Encaixe a variável `get global Resultado` para exibição. E na opção `Places` , responsável pela formatação da quantidade de números que teremos após a vírgula, arraste um bloco numérico, também da guia `Math` , e informe dentro dele a quantidade de casas decimais desejadas. No nosso caso, duas casas após a vírgula. Veja na figura seguinte o bloco finalizado.

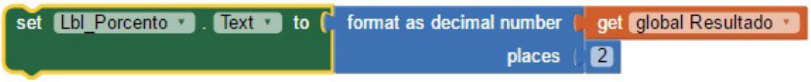

Figura 4.28: Exibição do resultado formatado com duas casas decimais

O ponto mais importante do nosso app, a decisão entre abastecer com etanol ou gasolina, será visto agora. Arraste mais um bloco `if` da guia `Controls` e configure-o para ter também uma opção `else` . Dentro desse bloco, encaixe um de comparação relacional **menor que** (com o símbolo `<`) da guia `Math` . No primeiro espaço, coloque a variável do resultado e, ao lado, encaixe um bloco numérico e insira nele o valor `0.7` .

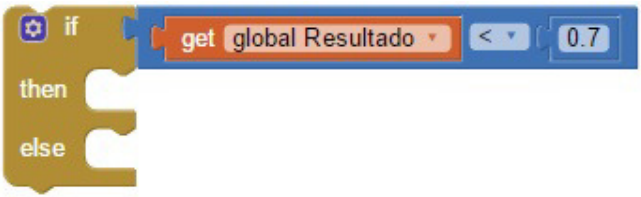

Figura 4.29: Verificando o resultado calculado

Caso a decisão seja verdadeira, devemos exibir no `Lbl_Resultado.Text` a informação **Etanol**; caso contrário, a mesma `Label` exibirá o texto **Gasolina**. Após o `then` , encaixe o bloco do `Lbl_Resultado.Text` e um bloco de texto e, dentro

deste, digite `Etanol`. Caso a decisão do bloco `if` não seja verdadeira, precisamos tratar a segunda opção, o `else`.

Novamente usaremos o bloco `Lbl_Resultado.Text` junto com um bloco de texto com o conteúdo `Gasolina`. Veja na figura a seguir o resultado:

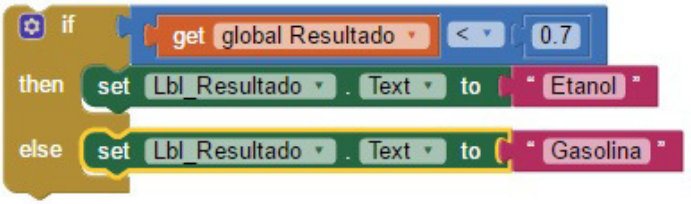

Figura 4.30: Bloco if com a verificação do resultado

Ao término dos blocos, para que o teclado do dispositivo não fique visível após a digitação dos valores, devemos inserir um bloco para escondê-lo. Logo, vamos usar um comando que chamará (`call`) a função de esconder o teclado. A função `call Txt_Gasolina.HideKeyboard` tem essa finalidade, e você encontra-a na guia do componente `Txt_Gasolina`.

Veja na figura a seguir como deverá ficar todo o bloco `Btn_Verifica`.

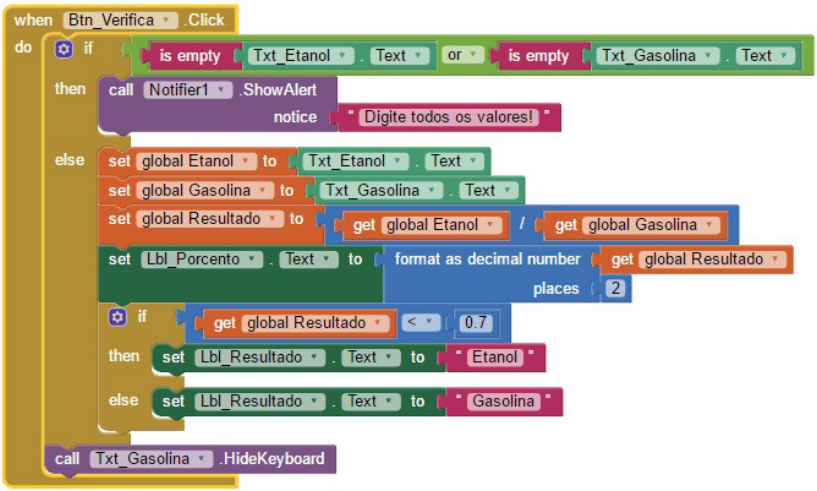
when Btn_Verifica .Click
do if is empty Txt_Etanol . Text or is empty Txt_Gasolina . Text
then call Notifier1 .ShowAlert
notice " Digite todos os valores! "
else set global Etanol to Txt_Etanol . Text
set global Gasolina to Txt_Gasolina . Text
set global Resultado to get global Etanol / get global Gasolina
set Lbl_Porcento . Text to format as decimal number get global Resultado
places 2
if get global Resultado < 0.7
then set Lbl_Resultado . Text to " Etanol "
else set Lbl_Resultado . Text to " Gasolina "
call Txt_Gasolina .HideKeyboard

Figura 4.31: Visualização do Btn_Verifica

Vamos supor que o usuário digitou algum valor errado e necessita apagar os dados da tela, ou mesmo que já tenha utilizado uma vez o aplicativo e queira realizar outros cálculos. É para essas ocasiões que preparamos o botão `Btn_Limpar`, que terá a função de apagar todos os dados visíveis nas `TextBoxs` usadas e deixar as `Labels` com os dados iniciais: `Lbl_Resultado` que receberá o valor `--`, e `Lbl_Porcento` que receberá o valor `0.00`.

Para executarmos o que foi descrito, insira o bloco `Btn_Limpar.Click` e, logo a seguir, coloque os blocos `set Lbl_Resultado to`, `set Lbl_Porcento to`, `set Txt_Etanol to` e `set Txt_Gasolina to`. Atribua um bloco de texto vazio para cada componente, depois acrescente os caracteres `--` para o bloco `Lbl_Resultado`, e para o `Lbl_Porcento`, os caracteres `0.00`.

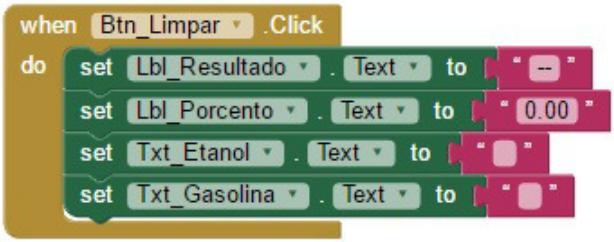
when Btn_Limpar .Click
do set Lbl_Resultado . Text to " -- "
set Lbl_Porcento . Text to " 0.00 "
set Txt_Etanol . Text to " "
set Txt_Gasolina . Text to " "

Figura 4.32: Bloco Limpar

Além de mover os valores demonstrados anteriormente, precisamos também apagar os valores contidos nas variáveis `set global etanol to` , `set global gasolina to` e `set global resultado to` e encaixe em cada uma um bloco **numérico** da guia `Math` com o valor zero. Confira na figura a seguir o resultado final do `Btn_Limpar` .

when Btn_Limpar .Click
do set Lbl_Resultado . Text to " -- "
set Lbl_Porcento . Text to " 0.00 "
set Txt_Etanol . Text to " "
set Txt_Gasolina . Text to " "
set global Etanol to 0
set global Gasolina to 0
set global Resultado to 0

Figura 4.33: Btn_Limpar

4.4 TELA DE INFORMAÇÃO

Precisamos de um botão para exibir uma segunda tela do aplicativo que informará ao usuário como é feito o cálculo. Vamos trabalhar com o botão `Btn_Informacoes` para realizar essa tarefa. Primeiramente, necessitamos inserir uma nova tela. Clique no botão `Add Screen` na parte superior do App Inventor.

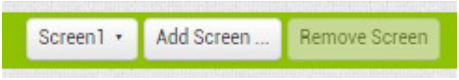

Figura 4.34: Inserindo uma nova Screen

Após o clique, será exibida a janela da figura seguinte, indicando que uma nova `Screen` (tela) será criada. Deixe o nome `Screen2` como padrão sugerido pelo App Inventor e clique no botão `OK`.

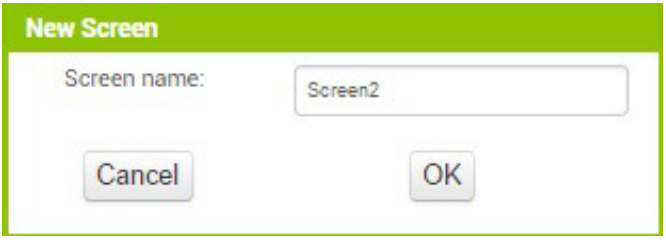

Figura 4.35: Criando uma nova Screen

Será apresentada uma nova tela vazia para nosso aplicativo. Vamos configurá-la para que apresente a visualização conforme a figura a seguir nos apresenta.

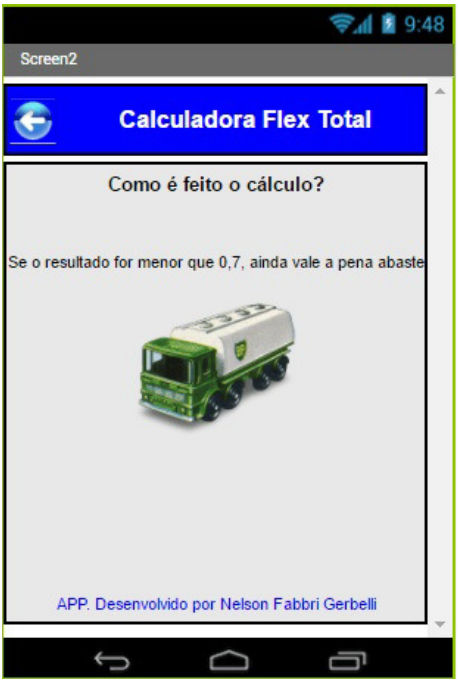
9:48
Screen2
Calculadora Flex Total
Como é feito o cálculo?
Se o resultado for menor que 0,7, ainda vale a pena abaste
APP. Desenvolvido por Nelson Fabbri Gerbelli

Figura 4.36: Tela de informações do app

Perceba que quase todos os componentes que vamos usar na `Screen2` já foram vistos e comentados anteriormente. Caso persista alguma dúvida, não evite em retomar a leitura.

Como construímos uma barra de título na `Screen1`, também vamos repetir o mesmo procedimento para a `Screen2`. Então, na propriedade `TitleVisible`, clique para desmarcar a opção padrão. Construiremos a barra de título com um botão para retornar à tela principal e uma `Label` para exibir propriamente o título do app.

A tabela a seguir apresenta os objetos que devemos inserir e as propriedades a serem alteradas. Vale lembrar que primeiramente

deverá ser inserido o componente `HorizontalArrangement` e, depois das configurações das suas propriedades, preenchê-lo com os componentes `Button e Label` .

Componente	Pallete	Propriedade	Alterar valor para
HorizontalArrangement	Layout	AlignVertical	Fill Parent
		BackgroundColor	Blue
		Height	50 pixels
		Width	Fill Parent
Button	User Interface	Height	35 pixels
		Width	35 pixels
		Image	Voltar.png
		Text	Apagar
		Renomear	Btn_Voltar
Label	User Interface	FontBold	Marcar
		FontSize	20
		Text	Calculadora Flex Total
		TextColor	White
		TextAlignment	Center
		Width	Fill Parent

Veja o resultado da barra de título da `Screen2` :

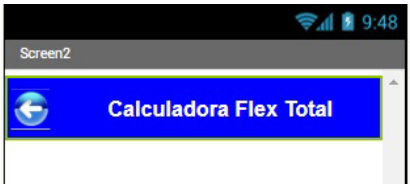

Figura 4.37: Barra de título

Vamos agora preparar a parte central da tela de informações. Abaixo da barra de título, coloque um componente `VerticalArrangement` da guia `Layout`. Como já falamos anteriormente, ele ajusta verticalmente os componentes que ficam em seu interior. Precisamos ajustar o seu comprimento através da propriedade `Width` para `Fill Parent`, pois assim o componente ocupará todo o espaço horizontal do dispositivo. Também queremos deixar centralizados todos os componentes que ainda vamos inserir nele, então, altere as propriedades `AlignVertical` e `AlignHorizontal` para a opção `Center`.

Agora precisamos pôr duas `Labels` no interior da `VerticalArrangement`. Insira uma abaixo da outra, e realize as alterações sugeridas na tabela.

Componente	Pallete	Propriedade	Alterar valor para
Label	User Interface	FontBold	Marcar
		FontSize	16
		Text	Como é feito o cálculo?
		TextAlignment	Center
		Height	50 pixels
		Width	Fill Parent

Componente	Pallete	Propriedade	Alterar valor para
Label	User Interface	FontBold	Marcar
		Text	Divida o valor do litro do etanol pelo da gasolina. Se o resultado for menor que 0.7, ainda vale a pena abastecer com etanol. Se for maior, opte pela gasolina.

Note na imagem a seguir o resultado da configuração das duas `Labels` inseridas, e observe que o texto explicativo de como é feito o cálculo não está sendo exibido por inteiro. Mas não se preocupe, pois, quando você estiver testando seu app, as informações aparecerão por inteiro em seu dispositivo.

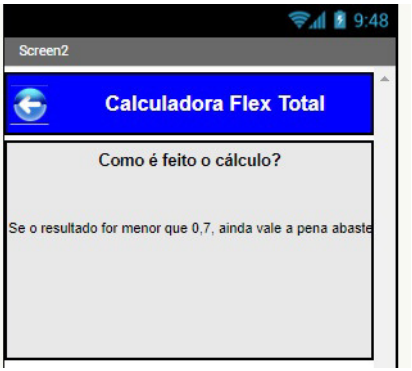

Figura 4.38: Labels configuradas

Para ilustrar a tela de informações, vamos inserir uma imagem com um caminhão tanque. Localize na guia `User Interface` um componente `Image`, responsável pela exibição de imagens, e coloque-o abaixo da `Label` que contém o texto da explicação dos cálculos. Para inserir uma imagem, vá até a propriedade `Picture` e realize o upload do arquivo `Leyland-Petrol-Tanker-icon.png`.

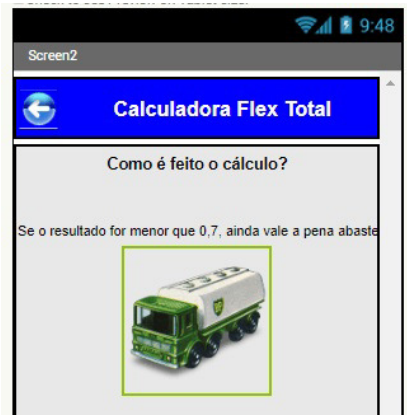

Figura 4.39: Imagem exibida

Para terminarmos a criação da tela de informações, só falta inserir duas `Labels` : a primeira servirá apenas para dar um espaçamento entre a imagem e a `Label` que vai conter a informação do desenvolvedor do app. Veja na tabela a seguir as propriedades que devem ser alteradas e seus novos valores.

Componente	Pallete	Propriedade	Alterar valor para
Label	User Interface	Height	90 pixels
		Text	Apagar
Label	User Interface	Width	Fill Parent
		Text	App desenvolvido por Nelson Fabbri Gerbelli
		TextColor	Blue

Com o design pronto, precisamos agora realizar a programação do botão voltar, que tem como finalidade fechar a tela de informações e retornar para a tela principal do aplicativo.

Na área dos blocos, selecione o componente `When Btn_Voltar.Click`. Agora precisamos de um comando para fechar a tela de informações. Para isso, acesse a opção `Control` na guia `Built-in` e selecione um `close screen`, posicionando-o dentro do `When Btn_Voltar.Click`. Veja na figura a seguir o resultado do `Btn_Voltar`.

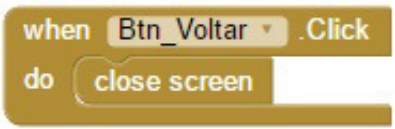

Figura 4.40: Btn_Voltar

Ainda na `Screen1`, é preciso realizar a chamada para a exibição da segunda `Screen`. Clique no botão `Screen2` no topo do App Inventor e selecione a `Screen1`, para retornar ao desenvolvimento da tela principal.

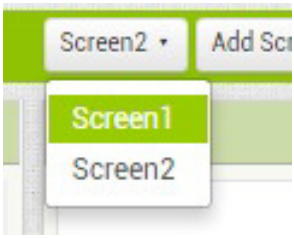

Figura 4.41: Acessando a Screen1

Selecione o componente `Btn_informações.Click` e arraste-o para a área de desenvolvimento. Agora na guia `Built-in`, na opção `Control` selecione um bloco `open another screen screenName`, que realiza a abertura e exibição da outra tela. Encaixe um bloco de texto, e digite internamente o nome da sua

segunda tela do aplicativo — no nosso caso, digite apenas `Screen2` . Atenção para as letras maiúsculas e minúsculas; é preciso digitar igual ao nome que foi dado durante a criação da segunda tela.

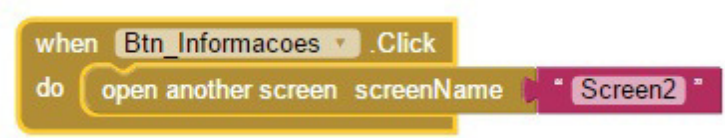

Figura 4.42: Btn_Informacoes para exibir a Screen2

4.5 TESTANDO O APLICATIVO

Para finalizarmos este capítulo, sugerimos que você emule seu aplicativo. Utilize uma das maneiras para emular, demonstradas no capítulo anterior.

Após a visualização da tela no emulador, digite os valores para o preço do litro do etanol e da gasolina e, ao final, clique no botão de calcular para ver o resultado de qual combustível é mais vantajoso. Não se esqueça de clicar no botão de informação para acessar a `Screen2` do app. As figuras a seguir exibem as telas do aplicativo em funcionamento.

Figura 4.43: Aplicativo em funcionamento

5554:<build>
5:22 PM
Calculadora Flex Total
Como é feito o cálculo?
Divida o valor do litro do etanol pelo da gasolina. Se o resultado for menor que 0,7, ainda vale a pena abastecer com etanol. Se for maior, opte pela gasolina.
APP. Desenvolvido por Nelson Fabbri Gerbelli
MENU

Figura 4.44: Exibindo a tela de informações

4.6 RESUMINDO

Neste capítulo, aprendemos a criar uma nova tela, utilizar as variáveis, realizar contas e a configurar a exibição de casas decimais. Vimos também decisões com o bloco `if/else` , como

limpar os dados já visualizados, e criamos um app que você poderá utilizar no dia a dia para decidir qual é o combustível que mais compensa quando for abastecer seu veículo flex.

No próximo capítulo, o leitor vai produzir um aplicativo que realizará a tradução de uma palavra, ou de um texto digitado ou falado, para um dos idiomas disponíveis. Além disso, será possível compartilhar a tradução por aplicativo externo, ou se desejar, ouvir o que foi traduzido através da leitura realizada pelo aplicativo.

Capítulo 5

TRADUTOR ONLINE

Neste capítulo, vamos desenvolver um aplicativo que realiza a tradução de textos do português para outros idiomas selecionados por você. Habilitaremos o reconhecimento de voz de seu dispositivo para informar o texto a ser traduzido pelo app. Além da tradução, você poderá compartilhar o resultado com seus contatos, ou simplesmente ouvir o texto que foi traduzido.

> Vá ao link de download dos projetos do livro (http://nelfabbri.com/appinventor/appinventor.zip) e baixe os arquivos que serão utilizados nesse aplicativo.

O app que desenvolveremos fará a tradução de uma palavra ou frase do português para o inglês, italiano, alemão ou espanhol, utilizando o componente de tradução `Yandex Translate` . O usuário poderá digitar as palavras a serem traduzidas na caixa de texto ou simplesmente usar o reconhecimento de voz que transformará o que foi falado em texto escrito. Para isso, veremos o componente `SpeechRecognizer` .

O aplicativo também possibilitará a leitura do texto que foi

traduzido através do componente `TextToSpeech`, e o compartilhamento do texto traduzido pelo aplicativo WhatsApp, por meio do componente `ActivityStarter`. Este transmitirá informações do nosso app para outro que está instalado em nosso dispositivo móvel. O aplicativo consumirá dados de internet, pois a tradução é realizada através de uma API externa.

> Uma API (*Application Programming Interface*) é um conjunto de padrões de programação disponíveis para que outros aplicativos utilize seus resultados.

5.1 APP TRADUTOR

Para desenvolver nosso tradutor, vamos iniciar um novo projeto no App Inventor. Clique na opção `Start New Project` e dê o nome de **Tradutor**.

Para o desenvolvimento do design do app, necessitaremos de cinco botões. Veja a seguir as funções de cada um:

1. O primeiro botão realizará o reconhecimento da voz do usuário;
2. O segundo vai efetivar a tradução;
3. O terceiro lerá o texto que foi traduzido;
4. O quarto botão limpará os textos da tela para realizar uma nova tradução;

5. O último botão vai possibilitar o compartilhamento da tradução com algum de seus contatos do WhatsApp.

Vamos conhecer o componente `ListPicker`. Ele é responsável por listar todos os idiomas que cadastraremos para realizar a tradução. Usaremos também `Labels` para exibir as mensagens e duas `TextBoxs`, que servirão para a digitação do texto a ser traduzido e para a exibição da tradução.

A tela do aplicativo é demonstrada na figura a seguir:

Figura 5.1: Tela do aplicativo tradutor online

Para começar, vamos realizar algumas modificações com a tela de fundo do nosso aplicativo. Na área `Components` do ambiente

de desenvolvimento do App Inventor, selecione a `Screen1` para habilitar as suas propriedades.

Figura 5.2: Habilitando as propriedades da Screen1

A tabela a seguir demonstra os componentes que deveremos inserir, o local onde os encontramos, as propriedades que deveremos alterar e os novos valores de cada uma.

Componente	Pallete	Propriedade	Alterar valor para
Screen1	- -	APPName	Tradutor
		BackgroundColor	Gray
		Icon	FALAR.png
		ScreenOrientation	Portrait
		TitleVisible	Desmarcar, pois criaremos a nossa barra de título personalizada.

A propriedade `APPName` indicará o nome do aplicativo que será exibido quando o leitor o instalar em seu dispositivo. A `BackgroundColor` altera a cor do fundo da tela, que aqui definimos como `Gray` (cinza). Na propriedade `Icon`, a imagem que utilizamos deverá ser exibida como ícone de seu aplicativo após a instalação em um dispositivo móvel. Faça o upload da imagem `FALAR.png`, conforme vimos em capítulos anteriores.

Definimos a propriedade `ScreenOrientation` para `Portrait`, para que a tela de seu aplicativo permaneça sempre na posição de retrato, ou seja, fique sempre em pé, independente de

deitarmos ou não nosso dispositivo. Já a propriedade `TitleVisible`, deve ser desmarcada para que o título padrão do App Inventor não apareça, pois desejamos criar uma área de título personalizada. Sua `Screen1` ficará conforme a figura:

Figura 5.3: Screen1

Vamos agora criar o título de nosso aplicativo, juntamente com um botão para a escolha da língua para a qual o seu texto deverá ser traduzido. Como desejamos ter dois componentes na mesma linha, torna-se necessário usar um componente que os agrupe, o `HorizontalArrangement`, e você poderá encontrá-lo na `Pallete` dos `Layouts`. Localize-o e o insira em sua `Screen1`. Note que ele já se posiciona na parte superior de sua tela.

A tabela a seguir mostra as propriedades que deverão ser alteradas e seus novos valores.

Componente	Pallete	Propriedade	Alterar valor para
HorizontalArrangement	Layout	AlignHorizontal	Right
		AlignVertical	Center
		BackGroundColor	Dark Gray
		Height	35 pixels
		Width	Fill Parent

A propriedade `AlignHorizontal` realiza o alinhamento dos componentes que ficarão em seu interior. Aqui, selecionamos o alinhamento para que todos os componentes fiquem alinhados à direita, por isso foi marcada a opção `Right` . Mas para um alinhamento referente ao seu espaço vertical dentro do `HorizontalArrangement` , a propriedade `AlignVertical` foi indicada como `Center` .

A propriedade `BackGroundColor` indica a cor de fundo da barra de título que está sendo desenvolvida, logo deverá ficar com uma cor diferente da usada no fundo. Escolhemos a cor `Dark Gray` (cinza escuro). Precisamos também alterar os tamanhos da altura e da largura que o `HorizontalArrangement` ocupará dentro da `Screen1` : a propriedade `Height` terá o valor de `35 pixels` e a `Width` terá a opção `Fill Parent` marcada, já que ocupará todo o espaço horizontal do seu dispositivo.

Dentro do componente `HorizontalArrangement1` , vamos inserir uma `Label` para exibir o título do aplicativo. Veja quais propriedades deveremos alterar:

Componente	Pallete	Propriedade	Alterar valor para
Label	User Interface	FontBold	Ativar
		FontSize	18
		Width	Fill Parent
		Text	Traduz Tudo
		TextAlign	Center
		TextColor	White

A propriedade `FontBold` habilitará o efeito de **negrito** e, em

`FontSize`, alteramos o **tamanho da fonte** para 18. Em `Width`, desejamos que o título ocupe todo o espaço interno do `HorizontalArrangement`. Já a propriedade `Text` mudará o nome de exibição, de `Text for Label1` para o texto sugerido: **Traduz Tudo**.

Como gostaríamos que o título ficasse centralizado na linha, alteramos a propriedade `TextAlign` para `Center`. A última formatação para a `Label` do título foi elegermos a cor do texto (`TextColor`), branco (`White`).

Ainda dentro da área de título no `HorizontalArrangement1`, devemos inserir o componente `ListPicker`, que exibirá os idiomas à disposição do usuário para a tradução. Insira o componente ao lado direito da `Label` que indica o título do aplicativo. A tabela seguinte mostra as propriedades e os valores do `ListPicker`:

Componente	Pallete	Propriedade	Alterar valor para
ListPicker	User Interface	ElementsFromString	Alemão, Espanhol, Inglês, Italiano
		Height	35 pixels
		Width	35 pixels
		Image	configuracao.png
		Text	Apagar todo o texto

A propriedade `ElementsFromString` exibe as possibilidades de idiomas selecionados para tradução. Fica a critério do leitor acrescentar ou excluir opções, mas, neste momento, sugiro manter as escolhas devido à programação dos blocos que veremos adiante.

Definimos o tamanho do botão, tanto sua altura (`Height`) como seu comprimento (`Width`), em `35 pixels` . A propriedade `Image` exibe uma imagem de fundo no componente `ListPicker` e, para isso, é preciso realizar o upload da imagem `configuracao.png` , conforme já demonstrado. Como queremos exibir apenas uma imagem no `ListPicker` , devemos deletar o texto de seu interior, apagando todo o conteúdo da propriedade `Text` .

Veja como deverá estar a visualização da barra de título, após as configurações feitas até aqui:

Figura 5.4: Barra de título

Precisamos inserir mais uma `Label` abaixo da área de título para informar ao usuário o que ele deverá fazer.

Componente	Pallete	Propriedade	Alterar valor para
Label	User Interface	Text	INSIRA O TEXTO PARA TRADUZIR
		TextColor	White

A figura a seguir exibe o resultado das alterações realizadas na `Label` .

Figura 5.5: Estágio atual da Screen1

Necessitamos de um espaço para a digitação do texto a ser traduzido. O componente utilizado para isso, como já vimos, é a `TextBox`. Insira-o imediatamente abaixo da `Label` com o texto **INSIRA O TEXTO PARA TRADUZIR**. Veja na seguinte tabela as configurações que a `TextBox` deverá sofrer.

Componente	Pallete	Propriedade	Alterar valor para
TextBox	User Interface	Height	80 Pixels
		Width	Fill Parent
		Hint	Digite seu texto
		Multiline	Deixar marcado
		Rename	Texto

Como já demonstramos as propriedades `Height` e `Width`, não repetiremos sua explicação, pois elas realizarão as mesmas funções. Já a propriedade `Hint` teve seu valor alterado para **Digite seu texto**. Essa informação será exibida dentro da `TextBox` quando seu aplicativo for executado e, ao escrever dentro dele, ela sumirá automaticamente, funcionando apenas como um lembrete do que deverá ser digitado em seu interior.

A propriedade `Multiline`, quando selecionada, permite que a frase contida no componente seja exibida em várias linhas.

Alteramos o nome da `TextBox` para `Texto`, pois futuramente será mais fácil identificar o componente pelo nome `Texto`. Veja o atual estágio em que a `Screen1` está em relação ao layout desenvolvido:

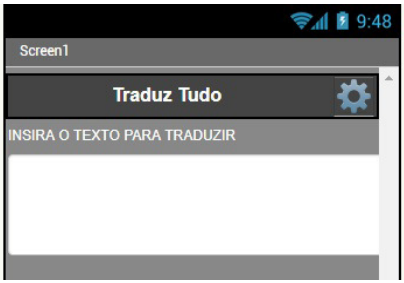

Figura 5.6: Layout atualizado

Para dar um espaçamento entre a `TextBox` que receberá o texto para ser traduzido e a próxima `Label` que indicará o texto já traduzido, precisamos inserir uma `Label` que terá apenas a função de separar esses dois componentes. Ela não conterá nenhuma informação, é apenas uma questão de ajuste de layout do app.

Insira-a logo abaixo da `TextBox` e realize as modificações em suas propriedades:

Componente	Pallete	Propriedade	Alterar valor para
Label	User Interface	Height	20 pixels
		Text	Apagar o texto

Agora, precisamos inserir a informação de que o texto apresentado logo abaixo é o resultado da tradução realizada. Para tanto, insira mais uma `Label` imediatamente abaixo da que

deixamos vazia e altere a propriedade que exibe o texto ao usuário (`Text`) para **TRADUÇÃO**. A propriedade que define a cor do texto (`TextColor`) deverá ser `White` .

Veja na figura o atual estágio da tela:

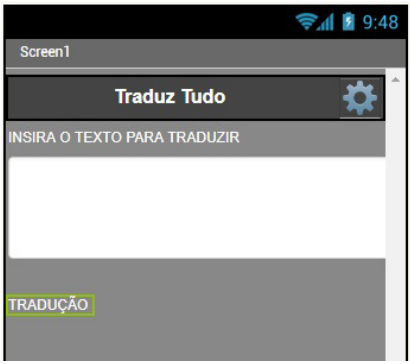

Figura 5.7: Layout atualizado

Necessitamos de um componente que exibirá a tradução realizada. Vamos utilizar o `TextBox` . Insira-o imediatamente abaixo da `Label` com o texto **Tradução** e realize as alterações em suas propriedades:

Componente	Pallete	Propriedade	Alterar valor para
TextBox	User Interface	BackgroundColor	Light Gray
		Enabled	ativar
		Height	80 Pixels
		Width	Fill Parent
		Hint	Texto Traduzido
		Multiline	Deixar marcado
		Rename	TRADUÇÃO

A única propriedade que está na tabela e ainda não foi apresentada é a `Enabled`. Quando ativada, ela desabilita a digitação em seu interior. Utilizaremos esse recurso, pois essa `TextBox` servirá apenas para exibir o resultado da tradução, não sendo necessário digitar nenhum conteúdo em seu interior.

Após as alterações, veja na figura a seguir como estará sua tela:

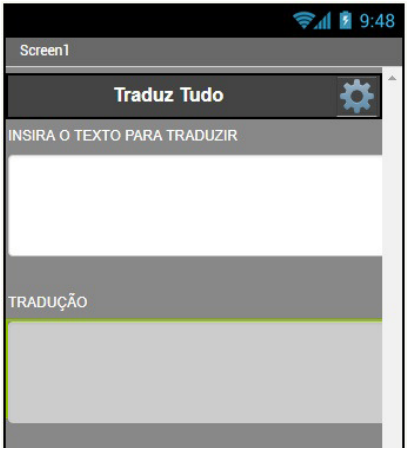

Figura 5.8: Layout atualizado

Novamente por questões de estética, vamos colocar mais uma `Label`, logo abaixo da `TextBox` do texto traduzido, para usar como um espaço entre os componentes. Veja na tabela a seguir as suas propriedades:

Componente	Pallete	Propriedade	Alterar valor para
Label	User Interface	Height	20 pixels
		Text	Deixar sem texto

Abaixo dessa última `Label` de espaçamento, vamos escrever no aplicativo o nome do seu desenvolvedor. Então, insira mais uma `Label` e realize as alterações em suas propriedades.

Componente	Pallete	Propriedade	Alterar valor para
Label	User Interface	FontBold	Selecionar
		Width	Fill Parent
		Text	Desenvolvido pelo Prof. Nelson Fabbri Gerbelli

Veja na figura a seguir como estará sua tela.

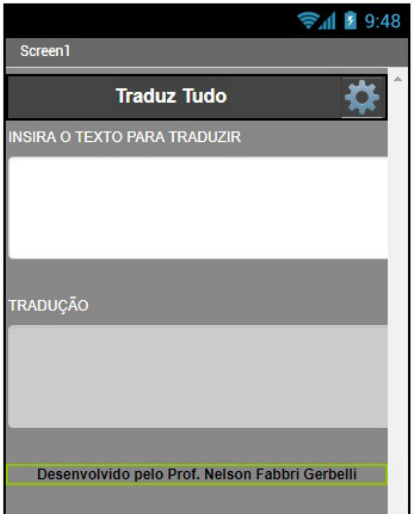

Figura 5.9: Layout atualizado com o nome do desenvolvedor

Abaixo do nome do desenvolvedor, prepararemos a área na qual os botões deverão ficar. Primeiramente, vamos inserir mais um componente que realizará sua organização horizontal, o já conhecido `HorizontalArrangement` , e realizar as seguintes

alterações:

Componente	Pallete	Propriedade	Alterar valor para
HorizontalArrangement	Layout	AlignVertical	Center
		BackgroundColor	Gray
		Width	Fill Parent

Para finalizarmos a criação da tela do nosso app, devemos inserir no interior do `HorizontalArrangement` cinco `Buttons`, cada um com sua função bem definida. Veja a tabela a seguir com as propriedades que devem ser alteradas e seus novos valores.

Componente	Pallete	Propriedade	Alterar valor para
Button	User Interface	Height	35 pixels
		Width	35 pixels
		Image	Microfone.png
		Text	Apagar o texto
		Rename	btn_reconhece_voz
Button	User Interface	Height	35 pixels
		Width	Fill Parent
		Text	TRADUZIR
		TextAlign	Center
		Rename	btn_Traduzir
Button	User Interface	Height	35 pixels
		Width	Fill Parent
		Text	Falar
		TextAlign	Center
		Rename	btn_Falar

Button	User Interface	Height	35 pixels
		Width	Fill Parent
		Text	LIMPAR
		TextAlign	Center
		Rename	btn_Limpar
Button	User Interface	Height	35 pixels
		Width	Fill Parent
		Image	whats.png
		TextAlign	Center
		Rename	btn_whats
		Text	Apagar o texto

Realize o upload das imagens indicadas na propriedade `Image` para a exibição no botão.

> Vale lembrar que as imagens `Microfone.png` e `whats.png`, assim como todas as outras usadas neste livro, estão disponíveis para download conforme indicado no começo do capítulo.

Devemos inserir também cinco componentes não visíveis que nos ajudarão a realizar as atividades propostas. São eles:

Objeto	Pallete	Função
Yandex Translate	Media	Uma API (Interface de Programação de Aplicativos) gratuita da empresa Yandex que está incorporada ao App Inventor, e é responsável pela tradução de mais de 70

		idiomas. Saiba mais através do endereço http://api.yandex.com/translate/.
TextToSpeech	Media	Componente que realiza a leitura do texto selecionado, ou seja, faz a vocalização do texto traduzido.
SpeechRecognizer	Media	Um componente de reconhecimento de voz para ouvir o que o usuário está falando e converter sua fala em texto. Utiliza o recurso de reconhecimento de fala do sistema Android.
ActivityStarter	Connectivity	Componente que realiza a abertura de outro aplicativo. No nosso caso, o nosso tradutor abrirá o aplicativo WhatsApp para envio e compartilhamento da tradução realizada.
Notifier	User Interface	Exibe uma mensagem de aviso ao usuário caso ele não possua o aplicativo **WhatsApp** instalado em seu dispositivo.

Veja como ficarão disponibilizados os componentes não visíveis em sua tela:

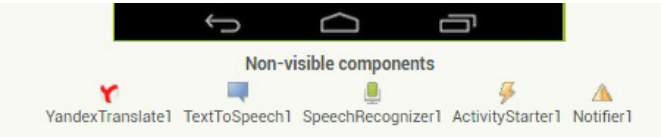

Figura 5.10: Componentes não visíveis

5.2 BLOCOS DO TRADUTOR

Com o design realizado e configurado, vamos clicar sobre o botão `Blocks` para começarmos a programar os blocos. Como teremos quatro possibilidades de idiomas para realizar a tradução, precisamos definir uma variável para receber e armazenar essa informação. Nesse caso, definiremos uma linguagem padrão para a tradução, pois se o usuário não selecionar nenhuma, o app vai traduzir automaticamente do português para o inglês.

Inicialize uma variável com o nome `idioma` e atribua um bloco de texto vazio. Dentro, escreva em letras minúsculas `pt-en`, indicando que vamos realizar uma tradução do idioma português para o inglês. Veja na próxima figura como deverá ficar definido o bloco da variável.

Cada língua possui seu próprio código, que é atribuído por letras ou números como sendo seu identificador. Em nosso aplicativo, vamos usar os seguintes códigos de línguas que indicam as traduções:

- **pt-en** — Português para inglês;
- **pt-de** — Português para alemão;
- **pt-es** — Português para espanhol;
- **pt-it** — Português para italiano.

Figura 5.11: Variável idioma configurada com valor padrão

Veremos a seguir a programação de cada botão de nosso aplicativo.

5.3 RECONHECIMENTO DE VOZ

Teremos um botão que fará o reconhecimento da voz do usuário e depois escreverá o texto na tela. Após clicar no botão `btn_reconhece_voz`, será ativado o reconhecimento de voz de

seu celular através do componente `SpeechRecognizer1` e este escreverá o que foi dito por você na `TextBox`.

Vamos programar esse botão. Para isso, na guia `Blocks`, localize o componente `btn_reconhece_voz`. Selecione o bloco `When btn_reconhece_voz.click` e arraste-o para a área `Viewer`.

Para o aplicativo reconhecer a sua fala, precisamos usar o bloco do componente `Call SpeechRecognizer1.GetText`. Localize-o em `Blocks` e encaixe-o no botão `btn_reconhece_voz`, pois somente assim será ativada a função de reconhecimento de voz do seu dispositivo. Veja na figura a seguir como deverá ficar o bloco `btn_reconhece_voz`.

Figura 5.12: Bloco de reconhecimento de voz

Agora que o aplicativo já identificou sua fala, devemos exibir o resultado no formato de texto dentro da `TextBox` que realizará a tradução. Vamos usar o evento que será executado após o recebimento do texto, o `AfterGettingText`. Então, inserimos o bloco `When SpeechRecognizer1.AfterGettingText` da guia de componentes. Veja como fica o bloco que receberá a fala:

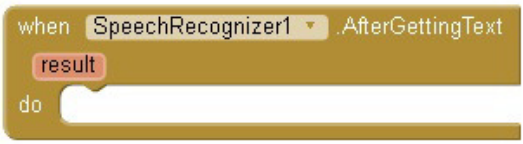

Figura 5.13: Bloco que recebe o texto falado

Para escrever o texto que foi ditado, necessitamos indicar o local onde ele será apresentado. Para a exibição, utilizaremos o componente `TextBox` que renomeamos como **Texto**. Insira um bloco `set Texto.Text to` da área de componentes no interior do `When SpeechRecognizer1.AfterGettingText`, pois é no componente `Texto.Text` que estará o texto a ser traduzido. Veja como está o desenvolvimento até o momento:

Figura 5.14: Preparando a TextBox para o recebimento do texto

Precisamos agora receber do aplicativo o texto para a exibição. O texto escrito a partir do que foi falado encontra-se na variável `result` do bloco `When SpeechRecognizer1.AfterGettingText`. Encontramos a variável `result` ao posicionarmos o ponteiro do mouse sobre a opção `result` no bloco principal e arrastando a opção `get result` para complementar o bloco `set Texto.Text to`. Veja o processo para encontrar a variável `get result` na figura seguinte.

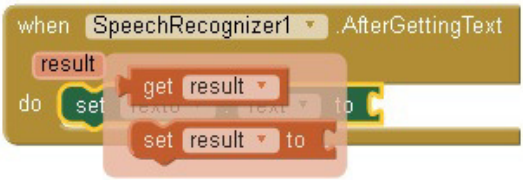

Figura 5.15: Selecionando a opção get result

Figura 5.16: Exibindo o texto falado

No reconhecimento de voz, devido ao barulho externo do seu ambiente, o aplicativo poderá não reconhecer sua fala perfeitamente. Nesse caso, será escrita alguma palavra errada, bastando apenas corrigi-la pelo teclado para não ocorrer erros na execução do aplicativo.

5.4 TRADUZINDO O TEXTO

Temos um botão que realiza a tradução do texto que já está visível na `TextBox` do aplicativo. Quando clicarmos nele para traduzir, vamos chamar a API do YandexTranslate e enviar todas as informações para que seja feita a tradução.

Vamos começar a programação inserindo um bloco do componente `btn_Traduzir.click` e, no seu interior, encaixar o bloco que realiza a chamada do componente `YandexTranslate1`, o `call YandexTranslate1.RequestTranslation`, para

requisitar a tradução. Dentro desse bloco, arraste a variável de idioma para informar de qual e para qual idioma o texto deverá ser traduzido, e encaixe-a na opção `languageToTranslateTo` . Vale lembrar que, na criação da variável idioma, nós definimos o valor **pt-en** como padrão.

Para finalizarmos esse bloco, informe o texto a ser traduzido, que se encontra no componente `TEXTO.text` , inserindo-o na opção `textToTranslate` . Note como deverá ficar seu bloco do botão `btn_Traduzir` na figura a seguir.

Figura 5.17: Bloco btn_Traduzir finalizado

Após a execução do bloco `call YandexTranslate1.RequestTranslation` , que enviou os dados para a tradução do texto, precisamos recebê-la e exibi-la no local correto. Para tanto, insira o bloco `When YandexTranslate1.GotTranslation` , que recebe o texto já traduzido. Porém, é preciso indicar o local para a exibição da tradução. Insira um bloco do componente `set TRADUCAO.Text to` para exibir a tradução que virá da API.

Para recebê-la, descanse o ponteiro do mouse sobre a opção `translation` e arraste o bloco `get translation` para completar a exibição:

Figura 5.18: Recebendo a tradução e exibindo-a no app

5.5 VOCALIZAÇÃO DA TRADUÇÃO

Temos um botão que poderá ler o texto que já foi traduzido. O componente responsável pela leitura é o `TextToSpeech`. Devemos chamar o `TextToSpeech` e indicar qual texto deverá ser lido.

Vamos inserir o bloco do componente `When btn_Falar.click`, pois a leitura será inicializada somente após o clique nele. Depois, arraste o bloco do componente `call TextToSpeech1.Speak` para iniciá-la.

Precisamos indicar qual texto deverá ser lido e encaixar na opção `message` o componente `TRADUCAO.TEXT`. A figura a seguir demonstra o botão de vocalização completo:

Figura 5.19: Botão que realiza a leitura do texto traduzido

Para realizar uma nova tradução, precisamos apagar os textos

exibidos na tela do app. Para isso, basta clicar no `btn_Limpar` que realizará tal tarefa.

Para realizarmos os blocos do botão limpar, é necessário encaixar espaços em branco em cada componente. Ou seja, adicionaremos uma caixa de texto sem preenchimento para as duas `TextBoxs` `set Texto.text to` e `set TRADUCAO.Text to` , conforme demonstrado na próxima imagem.

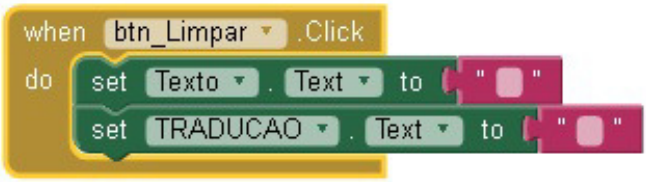

Figura 5.20: Botão que limpa os dados da tela

5.6 SELECIONANDO O IDIOMA PARA A TRADUÇÃO

Como nosso aplicativo poderá traduzir o texto para uma das línguas cadastradas no `ListPicker` , precisamos identificar qual foi a seleção do usuário. Quando o usuário clicar no ícone de configuração (`ListPicker1`), aparecerão as 4 opções já inseridas na tela de design (Alemão, Espanhol, Inglês e Italiano). Quando ele escolher o idioma desejado, o aplicativo deverá identificar qual índice do `ListPicker` foi selecionado.

Conseguimos identificar esse índice pela propriedade `SelectionIndex` do `ListPicker1` , e devemos guardar esse valor na variável `idioma` . A partir deste ponto, temos de realizar uma série de decisões para identificar qual língua foi escolhida. Se

for selecionada a primeira opção, teremos o índice 1; se a opção for Espanhol, o índice será o número 2, e assim sucessivamente.

Como o aplicativo também lerá a tradução feita, devemos realizar os ajustes necessários do idioma para a tradução e leitura do texto traduzido, pois se não o alterarmos, a leitura será feita com um sotaque. Por exemplo, para a leitura de um texto em alemão, estaríamos usando um leitor no idioma inglês, e não queremos isso.

Na prática, vamos inserir primeiramente um bloco indicando que, após a seleção de um idioma constante no `When ListPicker1.AfterPicking`, devemos realizar todas as tarefas descritas anteriormente. Em seu interior, vamos inserir a variável `set global idioma to` para receber o número do índice selecionado no `ListPicker1`. Ele é identificado através do bloco `ListPicker1.SelectionIndex`, que deverá ser posicionado na variável.

Figura 5.21: Variável recebendo o idioma selecionado

Agora que a variável já recebeu a identificação do idioma selecionado, devemos realizar as verificações para o ajuste da tradução através do bloco `if`. Insira um bloco de controle `if`, e nele encaixe um bloco de comparação com o sinal de igual, da guia `Built-in` na seção `Logic`. Dentro dele, é necessário inserir a variável que contém o número do idioma selecionado para ser comparado com o número do primeiro idioma que definimos

anteriormente.

Caso essa comparação seja verdadeira, faremos as configurações do código da língua para o app realizar a tradução. Insira um bloco de variável `set global idioma to` para receber o bloco de texto com o valor `pt-de` que traduzirá do português para o alemão. Para configurarmos o leitor com voz nesse idioma, necessitamos inserir os blocos `Language` e `Country` do `TextToSeech1` , e neles colocar um bloco de texto com o código `DEU` para a vocalização.

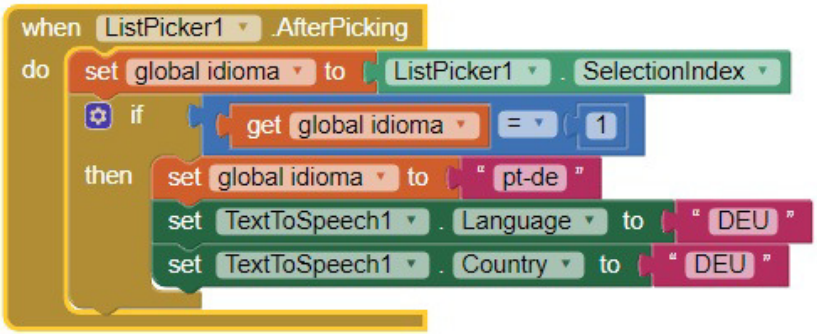
when ListPicker1 .AfterPicking
do set global idioma to ListPicker1 . SelectionIndex
if get global idioma = 1
then set global idioma to " pt-de "
set TextToSpeech1 . Language to " DEU "
set TextToSpeech1 . Country to " DEU "

Figura 5.22: Idioma alemão selecionado

A prática demonstrada anteriormente deverá ser realizada para os demais idiomas disponíveis. Veja na tabela a seguir os dados para a configuração dos idiomas e realize as configurações dos blocos faltantes.

Idioma	Código	Linguagem	País
Espanhol	pt-es	esp	ESP
Inglês	pt-en	en	USA
Italiano	pt-it	it	ITA

Após a inserção dos blocos, veja como deverá ficar seu bloco de seleção de idioma:

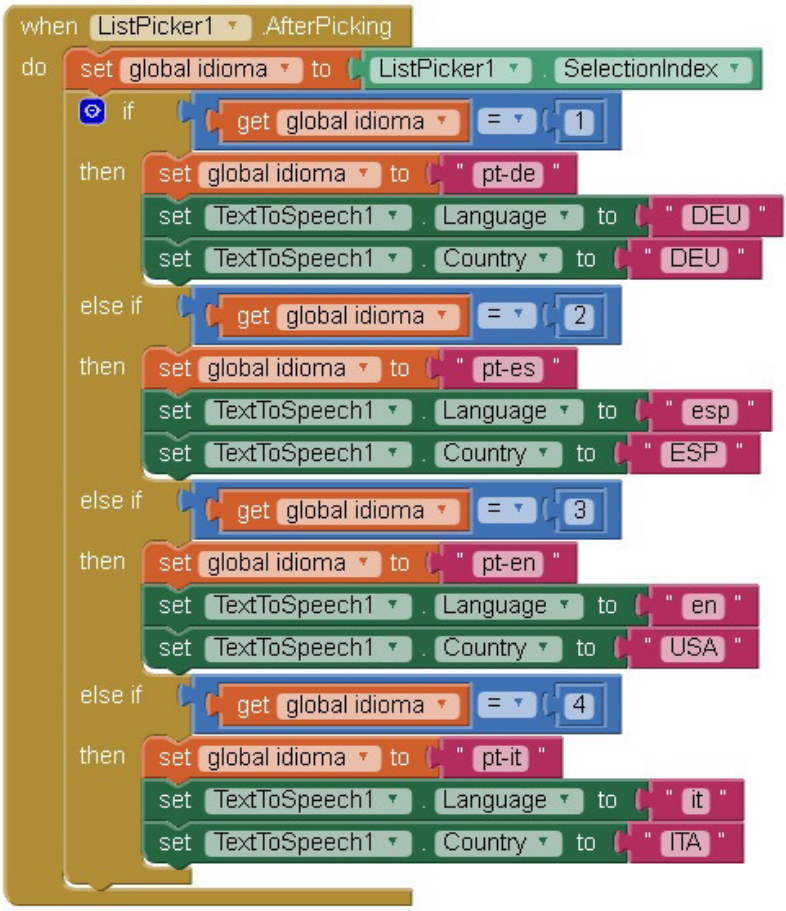
when ListPicker1 .AfterPicking
do set global idioma to ListPicker1 . SelectionIndex
if get global idioma = 1
then set global idioma to " pt-de "
set TextToSpeech1 . Language to " DEU "
set TextToSpeech1 . Country to " DEU "
else if get global idioma = 2
then set global idioma to " pt-es "
set TextToSpeech1 . Language to " esp "
set TextToSpeech1 . Country to " ESP "
else if get global idioma = 3
then set global idioma to " pt-en "
set TextToSpeech1 . Language to " en "
set TextToSpeech1 . Country to " USA "
else if get global idioma = 4
then set global idioma to " pt-it "
set TextToSpeech1 . Language to " it "
set TextToSpeech1 . Country to " ITA "

Figura 5.23: ListPicker que seleciona o idioma para tradução

Uma observação importante para esse momento é que, após inserir o bloco de controle `if`, temos de colocar os blocos de `else if`, que realizarão uma outra decisão caso a anterior não

seja verdadeira. Precisaremos testar as quatro opções de idioma possíveis, dependendo da escolha feita pelo usuário.

Para isso, clique no ícone de configuração ao lado esquerdo do bloco `if` (na cor azul) e arraste o bloco `else if` para dentro do controle:

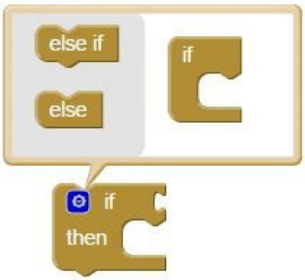

Figura 5.24: Botão de configuração do IF

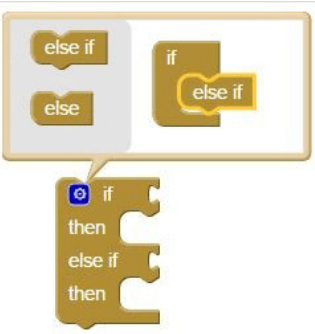

Figura 5.25: If com a opção ELSE IF

5.7 COMPARTILHANDO SUA TRADUÇÃO

O componente `ActivityStarter` permite que você combine aplicativos fazendo com que um inicie outras aplicações. É uma

maneira fundamental de ampliar os recursos do App Inventor, aproveitando outras ferramentas, sejam elas aplicativos criados com o App Inventor ou não. Eles também podem ser aplicativos como Câmera e Maps, que estão pré-instalados no dispositivo.

Como dissemos no início deste capítulo, teremos um botão que vai compartilhar a tradução com um de seus contatos do aplicativo externo WhatsApp. Como se trata do uso de um aplicativo que não foi desenvolvido por nós, o nosso deverá enviar informações para ele. Para essa tarefa, precisaremos realizar alguns ajustes no componente `ActivityStarter`, pois é ele que ativará o WhatsApp.

Esses ajustes ocorrerão quando o usuário clicar no botão `btn_whats`. Então, é necessário inserir um bloco do componente `When btn_whats.Click` e, no seu interior, vamos ter vários blocos de configuração do `ActivityStarter1`. A tabela a seguir exibe os blocos que deverão ser inseridos e seus valores.

Linha	Propriedade	Valor para digitação
1	Action	android.intent.action.SEND
2	ActivityPackage	com.whatsapp
3	ActivityClass	com.whatsapp.ContactPicker
4	DataType	text/plain
5	ExtraKey	Android.intent.extra.TEXT
6	ExtraValue	Bloco com o texto traduzido

Vamos entender cada **propriedade** do `ActivityStarter1` e seus **valores** que foram alterados:

1. Define a ação a ser realizada pelo seu aplicativo. Nesse caso,

gostaríamos de enviar os dados, por isso, utilizamos a opção `SEND`.

2. Indica o nome do aplicativo que queremos usar.

3. Indica qual a funcionalidade do aplicativo que desejamos usar. Aqui foi indicado que utilizaremos a opção de abrir a lista de contatos `ContactPicker`.

4. Indica o tipo de informação que vamos enviar para o aplicativo. Nesse caso, a informação é do tipo texto.

5. O ExtraKey representa qualquer informação adicional. Nesse caso, estamos dizendo que vamos usar o envio de um texto para o WhatsApp.

6. Indica qual texto será enviado. Em nosso caso, vamos encaminhar o conteúdo do componente que contém o texto traduzido, ou seja, o `TRADUCAO.TEXT`.

Veja como deverá ficar momentaneamente configurado o bloco do botão `btn_whats`.

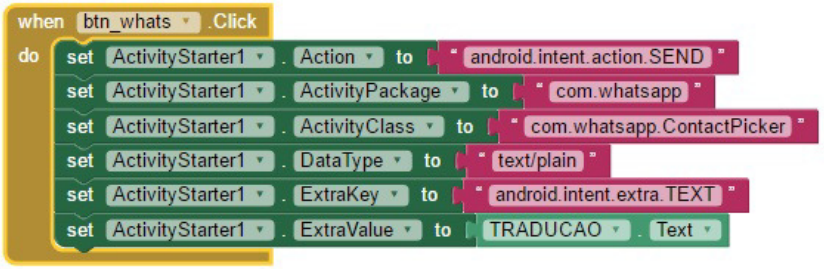
when btn_whats .Click
do
set ActivityStarter1 . Action to " android.intent.action.SEND "
set ActivityStarter1 . ActivityPackage to " com.whatsapp "
set ActivityStarter1 . ActivityClass to " com.whatsapp.ContactPicker "
set ActivityStarter1 . DataType to " text/plain "
set ActivityStarter1 . ExtraKey to " android.intent.extra.TEXT "
set ActivityStarter1 . ExtraValue to TRADUCAO . Text

Figura 5.26: Configurações do ActivityStarter

Após as configurações do `ActivityStarter1` que preparam

o seu app para enviar dados, teremos de verificar se o usuário tem esse aplicativo instalado em seu dispositivo. Para tanto, é necessário inserir um bloco `if` e realizar a verificação de sua existência.

Caso o WhatsApp não esteja instalado, informaremos ao usuário por meio de um bloco do `notifier` ; caso contrário, ativaremos a chamada do WhatsApp utilizando o comando `Call ActivityStarter1.StartActivity` . Veja a seguir como deverá ficar o bloco `if` com a verificação da instalação do WhatsApp.

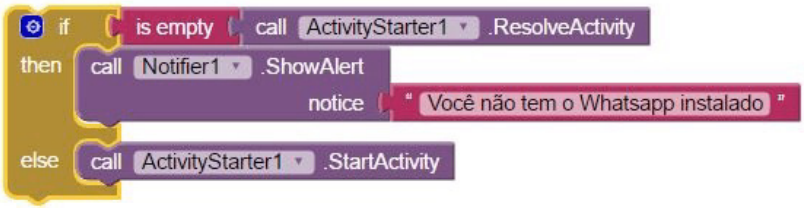

Figura 5.27: Verificando se o WhatsApp está instalado

A seguir, vemos uma imagem com o bloco completo do `btn_whats` .

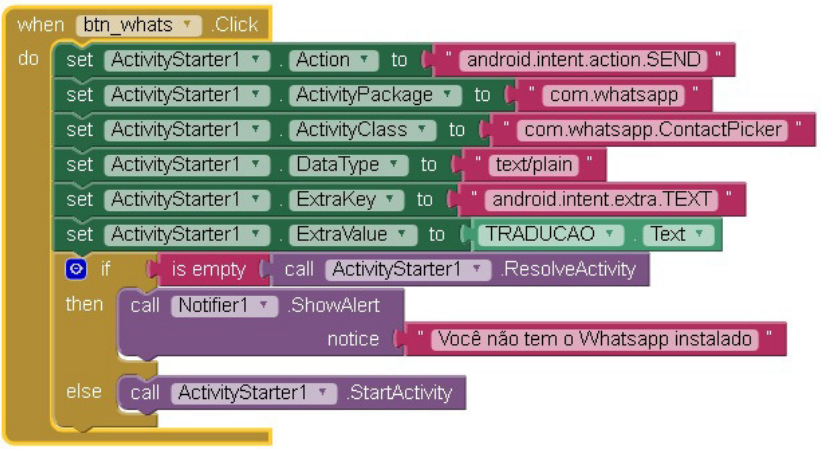
when btn_whats .Click
do
set ActivityStarter1 . Action to " android.intent.action.SEND "
set ActivityStarter1 . ActivityPackage to " com.whatsapp "
set ActivityStarter1 . ActivityClass to " com.whatsapp.ContactPicker "
set ActivityStarter1 . DataType to " text/plain "
set ActivityStarter1 . ExtraKey to " android.intent.extra.TEXT "
set ActivityStarter1 . ExtraValue to TRADUCAO . Text
if is empty call ActivityStarter1 .ResolveActivity
then call Notifier1 .ShowAlert
notice " Você não tem o Whatsapp instalado "
else call ActivityStarter1 .StartActivity

Figura 5.28: btn_whats completo

5.8 TESTANDO O APLICATIVO

Chegamos ao momento de testarmos o nosso tradutor. Clique para emular e instale-o no seu dispositivo físico, pois assim você poderá testar todos os recursos do aplicativo. A figura a seguir exibe a tela do reconhecimento de voz ativado pelo botão `btn_reconhece_voz` .

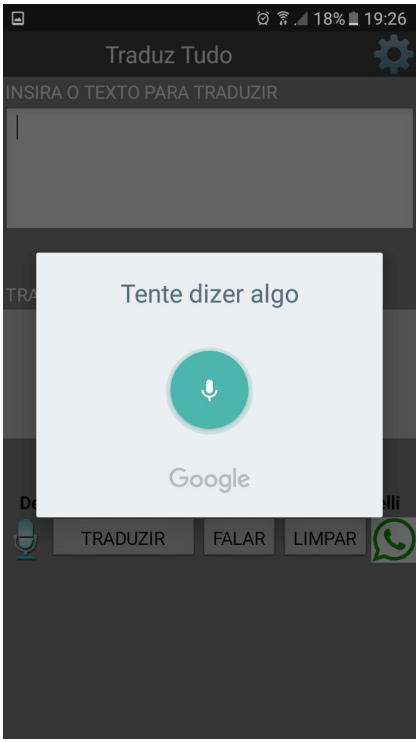

Figura 5.29: Reconhecimento de voz ativado

Após o usuário digitar ou falar um texto, será realizada uma tradução do português para o inglês, ao clicarmos no botão `btn_Traduzir` . Seu resultado pode ser visto a seguir.

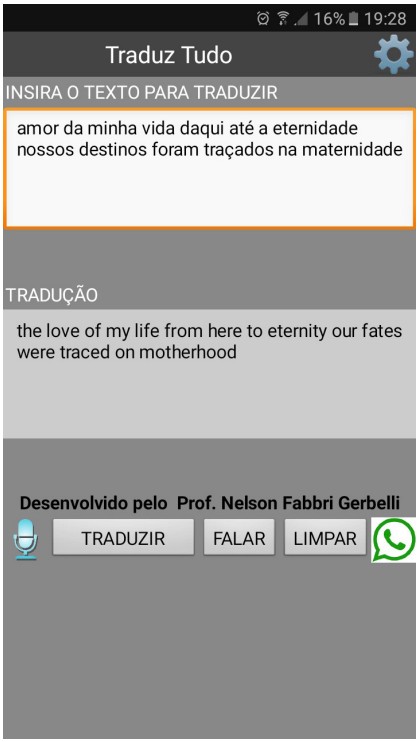
16% 19:28
Traduz Tudo
INSIRA O TEXTO PARA TRADUZIR
amor da minha vida daqui até a eternidade
nossos destinos foram traçados na maternidade
TRADUÇÃO
the love of my life from here to eternity our fates
were traced on motherhood
Desenvolvido pelo Prof. Nelson Fabbri Gerbelli
TRADUZIR
FALAR
LIMPAR

Figura 5.30: Tradução já realizada pelo app

Como a tradução é realizada por uma API, será necessário que o leitor possua em seu dispositivo móvel uma conexão com a internet; caso contrário, não será possível realizar a tradução.

Teste cada um dos botões que foram programados e, caso ocorra algum erro, volte para verificar os blocos do botão que o ocasionou, corrigindo-os se necessário. Uma dica importante em caso de erro é verificar os comandos que foram digitados. Veja se estão da maneira que indicamos, respeitando as letras maiúsculas e minúsculas, pois esse pequeno detalhe é uma das grandes causas de vários problemas na execução do aplicativo.

Deixo aqui uma sugestão e, ao mesmo tempo, um desafio para o leitor que concluiu o app de tradução: você poderá trocar ou inserir novas línguas conforme a sua necessidade.

5.9 RESUMINDO

No próximo capítulo, o leitor vai produzir um aplicativo que o ajudará a encontrar o local onde deixou seu carro estacionado. Utilizando o geolocalizador do dispositivo em conjunto com uma API da Google, poderemos traçar a rota para chegar até ele.

CAPÍTULO 6

ONDE ESTACIONEI?

Como toda solução nasce de um problema, desenvolvi este aplicativo pensando nas pessoas que, assim como eu, acabam por ter dificuldades de lembrar onde estacionaram seu carro e como encontrá-lo novamente. O app **Onde Estacionei?** nos ajudará traçando a rota de volta até o veículo. Este app também poderá ser útil em outras oportunidades, como voltar a um ponto de partida de um lugar pouco conhecido, geralmente em viagens.

O aplicativo funcionará recebendo o endereço e as coordenadas do satélite através do componente `LocationSensor` somente quando solicitado pelo usuário. Também armazenará as informações do local do estacionamento pelo componenete `TinyDB` . Quando desejarmos retornar ao ponto inicial, deveremos clicar no botão para localizar as atuais coordenadas do satélite, e solicitar que uma API exiba o mapa e trace a rota até o destino.

Por ser um componente que recebe valores do satélite, o `LocationSensor` funcionará apenas a céu aberto. Haverá consumo de dados de internet apenas para o uso do mapa, pois a rota é realizada através da API externa do Google Maps.

O leitor deverá ter instalado em seu dispositivo o aplicativo

Google Maps. Caso não o possua, vá até a *Play Store* e realize o download e a instalação do app *GPS e transporte público*.

6.1 INICIANDO O LAYOUT DO APP

Vamos criar um novo projeto no App Inventor, dando-lhe o nome de `Onde_Parei` . Antes de começarmos a desenvolver o layout do projeto, podemos verificar na figura a seguir como ficará a tela do nosso aplicativo.

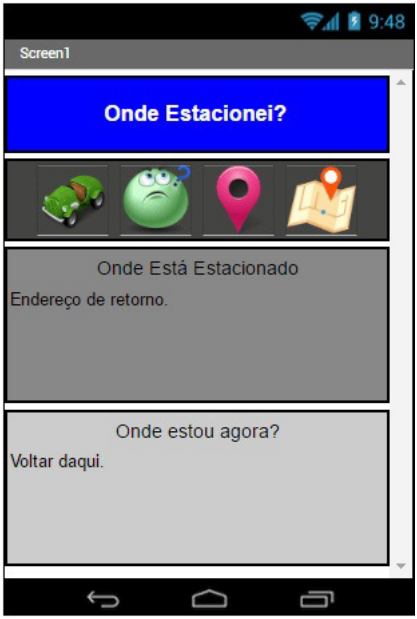

Figura 6.1: Tela do aplicativo Onde Estacionei?

Para o desenvolvimento dessa tela, vamos precisar de dois `HorizontalArrangement` . O primeiro servirá para a barra de títulos de seu aplicativo, e o segundo receberá os botões que terão as funções de localização do endereço do estacionamento. Teremos

algumas `Labels` que serão responsáveis pela exibição de informações e dois `VerticalArrangements` que farão parte da estética do aplicativo, servindo apenas para a organização das informações internas.

A tabela a seguir demonstra os componentes que deveremos inserir, o local onde os encontramos e as propriedades que devemos alterar. Vamos começar configurando a sua tela. Sem inserir nenhum componente ainda, realize as modificações:

Componente	Pallete	Propriedade	Alterar valor para
Screen	- -	AppName	Onde Estacionei?
		Icon	car.png
		TitleVisible	Desmarcar

Como já vimos em capítulos anteriores, alteramos o nome de exibição do aplicativo quando ele for instalado em `AppName`, realizamos o upload do ícone `car.png` e desabilitamos o título padrão da `Screen` para podermos criar nossa própria barra de título.

Para criar a nossa barra de título personalizada, vamos novamente utilizar o componente `HorizontalArrangement` para, em seu interior, inserir uma `Label`.

Componente	Pallete	Propriedade	Alterar valor para
HorizontalArrangement	Layout	AlignHorizontal	Center
		AlignVertical	Center
		BackgroundColor	Blue
		Height	60 pixels

		Width	Fill Parent
Label	User Interface	FontBold	Marcar
		FontSize	20
		Text	Onde Estacionei?
		TextColor	White

Para o componente `HorizontalArrangement` , realizamos o alinhamento de seu conteúdo ao centro, tanto na propriedade `AlignHorizontal` como na `AlignVertical` . Deixamos a cor de fundo selecionada como azul (`Blue`), e alteramos sua altura e seu comprimento horizontal.

Já com a `Label` que exibirá o título do aplicativo, mudamos o tamanho da fonte para `20` . Alteramos a propriedade `Text` para exibir a informação `Onde Estacionei?` e deixamos a cor do texto marcada como `White` .

A figura a seguir exibe a tela desenvolvida até o momento:

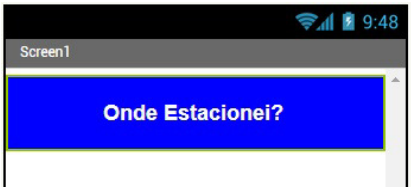

Figura 6.2: Estágio atual do desenvolvimento

Após configurar a barra de título, necessitamos de um `HorizontalArrangement` para organizar os quatro botões que realizarão todo o trabalho do app. São eles:

- **1º botão:** localiza suas coordenadas de estacionamento e salva em seu dispositivo.
- **2º botão:** exibe o endereço do local do estacionamento.
- **3º botão:** localiza o ponto de partida para retornar ao local de estacionamento.
- **4º botão:** traça a rota de retorno ao carro e exibe-a em um mapa.

Vamos preparar essa nova parte da `Screen` : insira logo abaixo da barra de título um componente `HorizontalArrangement` e configure-o conforme demonstrado na tabela a seguir. Após essas configurações, insira os quatro botões de trabalho no interior da `Screen` , não se esquecendo de configurá-los com os dados da tabela.

Componente	Pallete	Propriedade	Alterar valor para
HorizontalArrangement	Layout	AlignHorizontal	Center
		BackgroundColor	Dark Gray
		Width	Fill Parent
Button	User Interface	Height	60 pixels
		Width	60 pixels
		Image	save.png
		Text	apagar o texto
		Rename	Btn_Guardar
Button	User Interface	Height	60 pixels
		Width	60 pixels

		Image	onde.png
		Text	apagar o texto
		Rename	Btn_Exibir
Button	User Interface	Height	60 pixels
		Width	60 pixels
		Image	aqui.png
		Text	apagar o texto
		Rename	Btn_EstouAqui
Button	User Interface	Height	60 pixels
		Width	60 pixels
		Image	mapa.png
		Text	apagar o texto
		Rename	Btn_Chegar

Com o `HorizontalArrangement`, deixamos a propriedade `AlignHorizontal` marcada como `Center`, com a intenção de alinhar os botões, além de indicar que o componente utilizará todo o espaço horizontal. Para padronizar o tamanho dos botões, marcamos as propriedades `Height` e `Width` como `60` pixels.

Realize o upload das imagens baixadas. Temos uma imagem específica para identificar o que fará cada botão. Como queremos que neles apareçam apenas as figuras, apagamos o texto da propriedade `Text`. Alteramos também cada nome dos componentes `Button` para uma melhor identificação no momento da programação dos blocos.

A figura a seguir exibe a tela desenvolvida com os botões de

controle:

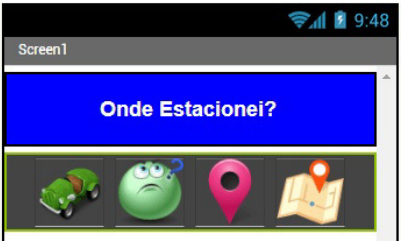

Figura 6.3: Estágio atual do desenvolvimento

Agora prepararemos um espaço para a exibição do local onde você estacionou seu veículo. Vamos utilizar um `VerticalArrangement` para organizar as informações em duas linhas. A primeira vai conter uma `Label` indicando o texto **Onde está estacionado**, e a segunda mostrará o endereço em que você estacionou.

Vamos lá. Insira abaixo do `HorizontalArrangement` dos botões um componente `VerticalArrangement` e, dentro dele, insira duas `Labels` . Realize as configurações indicadas na tabela:

Componente	Pallete	Propriedade	Alterar valor para
VerticalArrangement	Layout	BackgroundColor	Gray
		Heigth	30 percent
		Width	Fill Parent
Label	User Interface	FontBold	Marcar
		FontSize	18
		Width	Fill Parent

Componente	Pallete	Propriedade	Alterar valor para

		Text	Onde está estacionado
		TextAlignment	Center
Label	User Interface	FontSize	16
		Text	Endereço de retorno.
		Rename	Lbl_Carro

No `VerticalArrangement`, alteramos sua cor de fundo, e deixamos sua altura e seu comprimento horizontal definidos. Na primeira `Label`, deixamos a formatação em negrito, alteramos o tamanho da fonte, e seu comprimento ocupará todo o espaço da linha em que se encontra. Escrevemos o texto `Onde está estacionado` na propriedade `Text` que ficará centralizada na linha. Já para a segunda `Label`, alteramos o tamanho da fonte, mudamos o texto de exibição para `Endereço de retorno.` e também a renomeamos para `Lbl_Carro`.

Figura 6.4: Estágio atual do desenvolvimento

Da mesma maneira que inserimos o `VerticalArrangement` anterior com duas `Labels`, necessitamos repetir o mesmo

processo para exibir o endereço atual para retornar ao veículo estacionado. Insira outro componente abaixo do `VerticalArrangement` e, em seu interior, mais duas `Labels`, com as seguintes configurações:

Componente	Pallete	Propriedade	Alterar valor para
VerticalArrangement	Layout	BackgroundColor	Light Gray
		Heigth	30 percent
		Width	Fill Parent
Label	User Interface	FontBold	Marcar
		FontSize	18
		Width	Fill Parent
		Text	Onde estou agora?
		TextAlignment	Center
Label	User Interface	FontSize	16
		Text	Voltar daqui..
		Rename	Lbl_Estou

Veja a tela final do aplicativo:

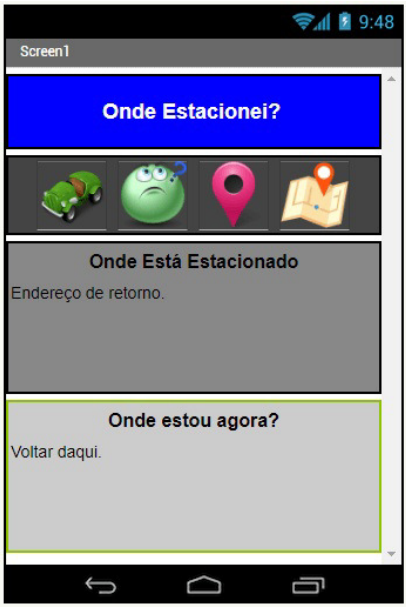

Figura 6.5: Tela final do aplicativo

A seguir, inseriremos os componentes não visíveis que farão parte do projeto:

Componente	Pallete	Propriedade	Alterar valor para
LocationSensor	Sensors	Enabled	Desmarcar
		Rename	Location_Parado

O componente `LocationSensor` , encontrado na `Pallete Sensors` , identifica a latitude e a longitude do satélite, bem como o endereço correspondente a essas coordenadas. Após sua inserção na `Screen` , desmarcamos a sua propriedade com `Enabled` , pois ele só será ativado quando houver uma solicitação do usuário. Alteramos também seu nome para `Location_Parado` para melhor identificação no momento em que estivermos trabalhando

com os blocos de programação.

Para armazenar as informações do local do estacionamento e para que elas fiquem disponíveis mesmo após o encerramento da execução do app, necessitamos de um componente da guia `Storage` : o `TinyDB` . Basta inseri-lo e nenhuma alteração nas propriedades será necessária.

Insira também o componente não visível `Notifier` , responsável por transmitir ao usuário a solicitação se queremos ou não salvar o endereço do estacionamento. Não se preocupe ainda com a resolução lógica do aplicativo, pois mais adiante veremos os procedimentos passo a passo.

Agora, necessitamos de um componente que realize a conexão do nosso aplicativo com o app *Google Maps*. Insira um componente `ActivityStarter` da guia `Connectivity` , e realize as seguintes alterações nas propriedades:

Linha	Propriedade	Alterar valor para
1	action	android.intent.action.VIEW
2	Activity Class	com.google.android.maps.MapsActivity
3	Activity Package	com.google.android.apps.maps

1. Define a ação a ser realizada pelo seu aplicativo. Nesse caso, gostaríamos de ter uma visão das informações, por isso utilizamos a opção `VIEW` .

2. Indica o nome do aplicativo que queremos utilizar, ou seja, o `MapsActivity` .

3. Indica qual a funcionalidade do aplicativo que desejamos

usar. Aqui indicamos que utilizaremos a opção de exibição do mapa, através da `maps`.

Da mesma maneira que inserimos o componente `Location_Parado`, deveremos inserir mais um `LocationSensor`, agora usando-o para a identificação do local de retorno. Após sua inclusão na `Screen`, altere os seus valores conforme demonstra a tabela:

Componente	Pallete	Propriedade	Alterar valor para
LocationSensor	Sensors	Enabled	Desmarcar
		Rename	Location_EstouAqui

A figura a seguir exibe os componentes não visíveis inseridos na área de design:

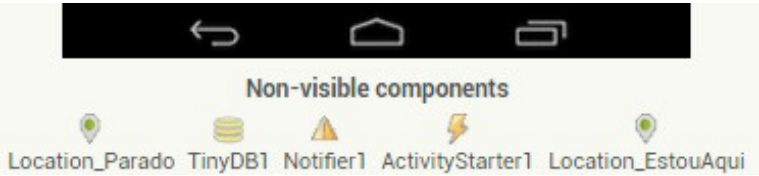

Figura 6.6: Componentes não visíveis

6.2 IDENTIFICANDO A LATITUDE E A LONGITUDE DO ESTACIONAMENTO

Para identificarmos o local em que estacionamos o veículo, precisamos armazenar em variáveis as coordenadas da latitude e da longitude logo que estacionarmos o carro. Você consegue identificá-las, utilizando o componente `LocationSensor`. Lembre-se de que ele foi renomeado para `Location_Parado`.

O componente `LocationSensor` ativa a geolocalização em seu dispositivo diretamente do satélite, não necessitando de uso de dados da internet. Porém, é preciso que o leitor tenha esse recurso em seu dispositivo.

Note também que desmarcamos a propriedade `Enabled` do componente `LocationSensor`, pois ele será habilitado para receber os dados da sua localização somente quando clicarmos no botão `Btn_Guardar`.

Primeiramente precisamos criar as variáveis para guardar as coordenadas de origem e também do ponto final, para quando quisermos voltar para encontrar o veículo estacionado. Inicialize quatro variáveis da guia `Built-in` e nomeie-as como `Latitude_car`, `Longitude_car`, `Latitude_atual` e `Longitude_atual`, em que as `Latitude_car` e `Longitude_car` indicarão o local onde você estacionou o veículo, e as `Latitude_atual` e `Longitude_atual` indicarão o ponto do retorno.

Deixe as quatro variáveis criadas como numéricas; elas só poderão receber números. Para isso, encaixe um bloco numérico da guia `Math` em cada uma. Em caso de dúvidas quanto à criação de variáveis, retome a leitura do capítulo *Uma simples calculadora*.

initialize global Latitude_car to 0
initialize global Longitude_car to 0
initialize global Latitude_atual to 0
initialize global Longitude_atual to 0

Figura 6.7: Variáveis do aplicativo

Precisamos ativar o componente `LocationSensor` para que ele receba a latitude e a longitude de onde acabamos de estacionar. Conseguimos isso apenas alterando a propriedade `Enabled` de `false` para `true`. Isso acontecerá quando o usuário clicar no `Btn_Guardar`.

Para que isso aconteça, insira um bloco `When Btn_Guardar.Click` na sua área `Viewer` e, dentro desse bloco, colocaremos um comando para ativar o localizador. Selecione o comando `set Location_Parado.Enable to` e, logo na sequência, encaixe um bloco lógico `true`. A figura a seguir exibe o `Btn_Guardar` com o localizador ativado.

Figura 6.8: Btn_Guardar ativando o geolocalizador

Quando ativarmos o geolocalizador, ele receberá as coordenadas e também o endereço do local onde estamos

estacionando. Devemos exibir o endereço em uma `Label` e guardar as coordenadas nas variáveis. Porém, antes de guardar os valores, precisamos ter certeza de que o `LocationSensor` já recebeu os valores do satélite, pois às vezes demora alguns segundos para essa tarefa ser executada.

Como não temos como saber o tempo exato que isso tomará, precisamos verificar constantemente. Teremos uma variável para guardar a informação assim que o endereço for exibido. Se essa variável de controle estiver marcada como verdadeira, podemos continuar e armazenar os dados nas variáveis `Latitude_car` e `Longitude_car` ; caso contrário, necessitaremos aguardar mais um tempo.

Precisamos criar a variável que verifica se o endereço foi recebido. Crie mais uma variável, atribua o nome de `verifica` e o valor `false` , pois só vamos continuar quando a variável tiver um valor verdadeiro `true` .

Figura 6.9: Criação da variável verifica

O componente que recebe as informações da localização que devemos inserir na área `Viewer` é o `when Location_Parado.LocationChanged` . A figura a seguir mostra o bloco para recebimento da latitude e longitude.

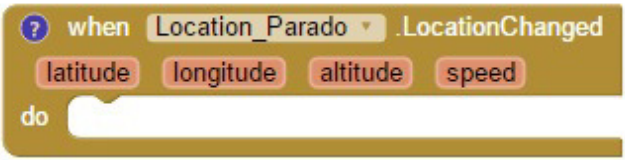

Figura 6.10: Recebendo os dados do localizador

Dentro desse bloco, necessitamos atribuir o endereço recebido pelo localizador para a `Lbl_Carro`. Primeiramente insira um `set Lbl_Carro.Text to` para receber o valor do endereço que está em `Location_Parado.CurrentAddress`, que deverá ser encaixado no `Llb_Carro`. Veja a seguir o bloco que exibe o endereço recebido pelo localizador.

Figura 6.11: Exibindo o endereço na Lbl_Carro

Após a exibição do endereço, devemos marcar a variável `verifica` como `true`, e assim guardar as informações nas variáveis. Insira um bloco para alterar o valor da variável. A próxima imagem exibe o bloco `when Location_Parado.LocationChanged` até o momento.

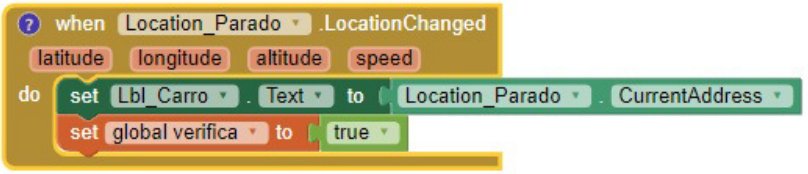

Figura 6.12: Recebendo todos os dados do localizador

Precisamos sempre confirmar se a variável `verifica` está com o valor `true`, pois é a partir desse ponto que armazenaremos os dados nas `Latitude_car` e `Longitude_car`. Existe um bloco que realiza uma verificação constante, o `While`. Sempre após esse bloco, existe uma comparação e, se o resultado dela for verdadeiro, será dada sequência aos comandos. A figura a seguir exibe o bloco `While test do` que deverá ser inserido:

Figura 6.13: Bloco While

Note que, após o comando `While`, existe uma opção de `test`, e é nesse espaço que deveremos testar se a variável `verifica` está com o valor verdadeiro. Insira um bloco lógico de comparação da guia `Built-in` logo após a opção `test`. Dentro desse bloco, insira `set global verifica` e, após o símbolo de igualdade, encaixe um bloco lógico `true`.

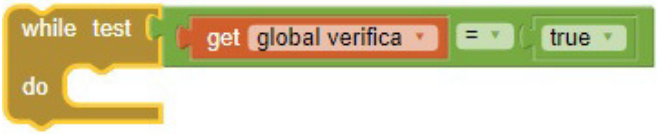

Figura 6.14: Bloco While com o teste de verificação

Quando o teste anterior for verdadeiro, precisaremos alterar o valor da variável `verifica` para `false`, para futura utilização do aplicativo. As variáveis `Latitude_car` e `Longitude_car` deverão receber os valores referentes às coordenadas do estacionamento do veículo, e é neste ponto que isso acontece.

Insira as variáveis `set global Latitude_car to` e `set global Longitude_car to` da guia `Built-in`. Para a variável que armazena a **latitude**, selecione a opção `get latitude` apenas posicionando o ponteiro do mouse sobre a opção e arrastando-a para encaixar na variável. Repita o mesmo procedimento para o encaixe da longitude na variável `Longitude_car`.

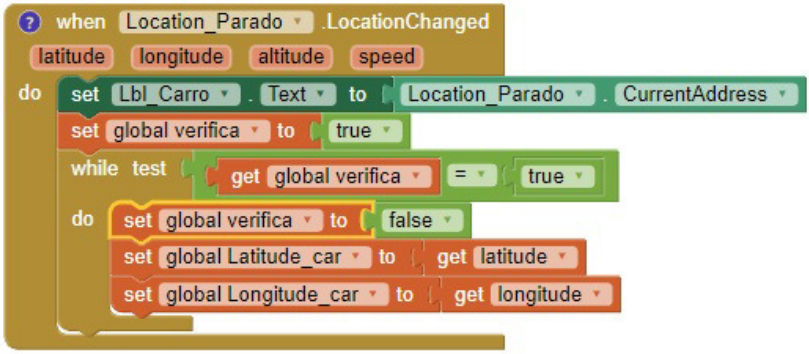
when Location_Parado .LocationChanged
latitude longitude altitude speed
do set Lbl_Carro . Text to Location_Parado . CurrentAddress
set global verifica to true
while test get global verifica = true
do set global verifica to false
set global Latitude_car to get latitude
set global Longitude_car to get longitude

Figura 6.15: Recebendo os dados do localizador

Antes de realmente salvarmos a localização do carro, é preciso perguntar ao usuário do aplicativo se ele realmente deseja guardar os dados. Caso responda que **Sim**, efetivaremos a gravação do local; mas se o usuário responder **Não** ou **Cancelar**, nada será executado.

6.3 ARMAZENANDO A LATITUDE E A LONGITUDE

Necessitamos criar uma caixa de diálogo com a pergunta **Salvar endereço atual?**, e os botões de resposta **Sim**, **Não** e **Cancelar**. Insira um bloco do componente

`Notifier.ShowChooseDialog` para realizar essa tarefa.

Note as opções de encaixe: `message`, `title`, `button1Text`, `button2Text` e `cancelable`. Nelas deverão ser encaixados blocos de texto com as seguintes informações: em `message`, digite **Salvar endereço atual?**. Em `title`, indicando o que será exibido na caixa de mensagem, digite **Atualizar?**. Já nos `button1Text` e `button2Text`, insira **Sim** e **Não**, respectivamente, para nomear os botões de resposta nos quais o usuário deverá clicar.

Na opção `cancelable`, já temos um bloco lógico definido como `true`. Ou seja, o botão de **cancelar** já vem configurado para exibição, mas, caso não queira que ele apareça, altere seu valor para `false`. Veja na figura como ficará o bloco do botão `when Location_Parado.LocationChanged` quando finalizado:

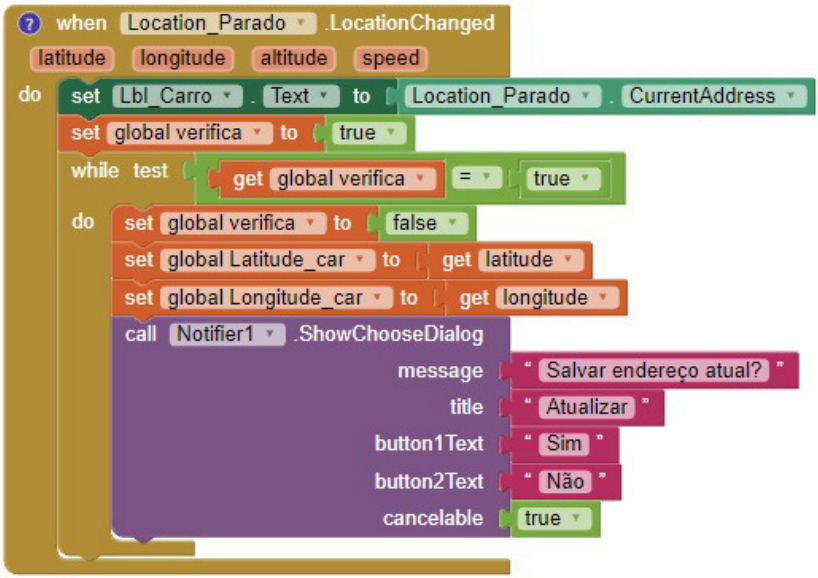
when Location_Parado .LocationChanged
latitude longitude altitude speed
do set Lbl_Carro . Text to Location_Parado . CurrentAddress
set global verifica to true
while test get global verifica = true
do set global verifica to false
set global Latitude_car to get latitude
set global Longitude_car to get longitude
call Notifier1 .ShowChooseDialog
message " Salvar endereço atual? "
title " Atualizar "
button1Text " Sim "
button2Text " Não "
cancelable true

Figura 6.16: Location_Parado finalizado

Ainda não configuramos nenhuma ação no clique dos botões do `Notifier`, que deseja saber se vamos salvar ou não as informações de parada. Para identificar a resposta do usuário, precisamos inserir o componente `When Notifier1.AfterChoosing`. Dentro do `Notifier`, colocaremos a decisão responsável por verificar se o usuário clicou no botão **Sim**.

Para isso, insira um bloco de controle `if` e, logo em seguida, selecione um bloco de texto que realize a comparação do botão clicado com o que virá do aplicativo. Para receber a escolha do usuário, posicione o ponteiro do mouse sobre `choice` e selecione a opção que surgirá `get choice` para, logo após, encaixar a decisão `compare texts`.

Finalmente, insira uma caixa de texto e, dentro dela, o nome do botão que você deseja tratar. Nesse caso, queremos programar o botão **Sim**, que gravará os dados da localização do usuário. A próxima figura exibe como fica a configuração do bloco `if`.

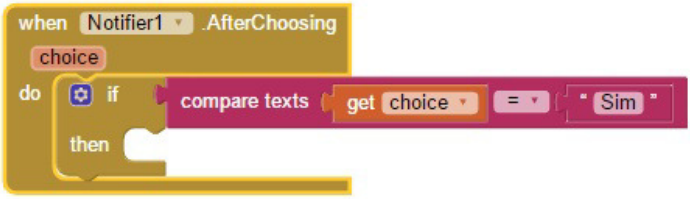

Figura 6.17: Verificando a escolha do usuário

Como é necessário guardar os valores da latitude, da longitude e do endereço do veículo estacionado, usaremos o componente `TinyDB`. Com ele, é possível armazenar dados e posteriormente recuperá-los. Essa é uma solução simples para armazenamento de

informações, mas atende exatamente ao que desejamos para o aplicativo.

O bloco `Call TinyDB1.StoreValue`, encontrado na guia de `Blocks` ao clicar sobre o componente `TinyDB1`, armazenará no dispositivo a `tag` (um nome de variável) e o valor através da opção `valueToStore`. Teremos de guardar o endereço, latitude e longitude do local onde estacionamos o veículo. Um único componente `TinyDB` pode armazenar várias informações, basta ir criando várias `tags` diferentes.

Incluiremos um bloco `Call TinyDB1.StoreValue` logo após o comando `then` do bloco `if`. Na opção `tag`, devemos colocar um bloco de texto com o nome da variável que armazenará o conteúdo. Na opção `valueToStore`, encaixe a `Label` que exibe o endereço recebido pelo componente `LocationSensor`, ou seja, a `Lbl_Carro.text`. A figura exibe o bloco que grava o endereço no componente `TinyDB1`:

Figura 6.18: Salvando o endereço no TinyDB1

Necessitamos salvar também a latitude e a longitude. Vamos usar o mesmo componente `TinyDB1` para armazenar esses dados, porém em `tags` diferentes.

Vamos inserir mais dois blocos do `Call TinyDB1.StoreValue` para realizar essa tarefa. No primeiro, na

opção `tag` , insira um bloco de texto e, dentro dele, defina o nome da variável `latitude` . Na opção `valueToStore` , encaixe a variável `get global Latitude_car` . Repita o procedimento com os dados da longitude, pois devemos salvar também sua coordenada.

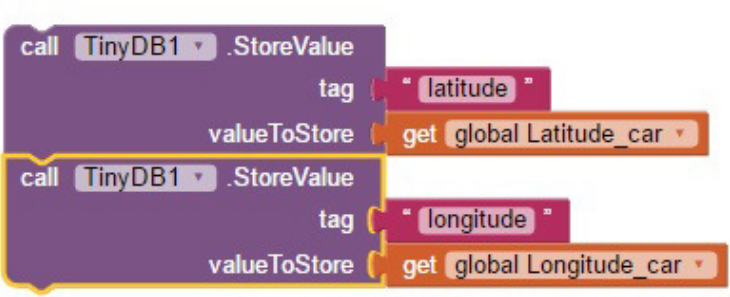

Figura 6.19: Salvando a latitude e a longitude no TinyDB1

Após salvar os dados em seu dispositivo, precisamos exibir uma notificação de que tudo ocorreu bem para o usuário. Prepare um bloco `Notifier.ShowAlert` para exibir um texto com a informação **Salvo com sucesso**. A próxima figura demonstra essa tarefa.

Figura 6.20: Exibindo a confirmação

Já utilizamos o componente de localização e precisamos desabilitá-lo. Caso não o façamos, o componente ficará buscando a cada momento a posição do usuário, mesmo não havendo mais

necessidade, pois já estacionamos o veículo e armazenamos os dados. Então, selecione um bloco `set Location_Parado.Enabled to` e encaixe um bloco lógico `false`, conforme a figura:

Figura 6.21: Desabilitando o localizador

A próxima figura exibe o bloco completo do `Notifier.AfterChoosing`:

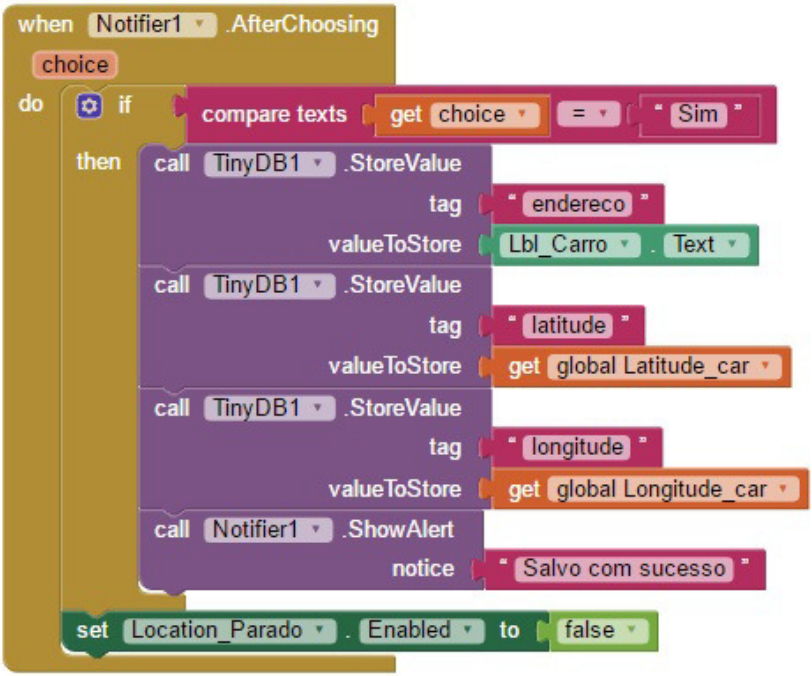

Figura 6.22: Bloco Notifier.AfterChoosing completo

6.4 EXIBINDO O ENDEREÇO DO ESTACIONAMENTO

Para rever o endereço do veículo que está gravado em seu app, bastar clicar no `Btn_Exibir` que as informações salvas no `TinyDB1` serão recuperadas e mostradas na `Label` de nome `Lbl_Carro.text`. Os valores da latitude e longitude não serão exibidos para o usuário e deverão ficar armazenados nas variáveis `Latitude_car` e `Longitude_car`.

Vamos inserir primeiramente um `When Btn_Exibir.Click` para, na sequência, incluir um componente `set Lbl_Carro.text to` para receber o endereço do `TinyDB1`. Para recuperar o endereço, inclua um componente `call TinyDB1.GetValue`.

Na opção `tag`, encaixe um bloco de texto e digite o nome da `tag` que contém o endereço salvo anteriormente. Nesse caso, digite apenas **endereco**. Na opção `valueIfTagNotThere`, coloque uma caixa de texto vazia. A sua função é de não exibir nada na `Label` se a tag `endereco` não existir. A figura a seguir demonstra o bloco que exibe o endereço recuperado do `TinyDB1`.

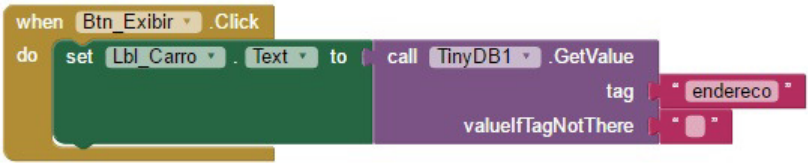

Figura 6.23: Recuperando o endereço do TinyDB1

Como até agora exibimos apenas o endereço, falta recuperar sua latitude e sua longitude. Como as salvamos no TinyDB1, devemos atribuir seus valores às variáveis `Latitude_car` e

`Longitude_car` .

Insira o bloco da variável `set global Latitude_car to` para receber o valor e, na sequência, encaixe um componente `call TinyDB1.GetValue` com o texto `latitude` na opção da `tag` . Em `valueIfTagNotThere` , coloque também uma caixa de texto vazia como demonstrado anteriormente. Necessitamos repetir esse processo para a variável `Longitude_car` .

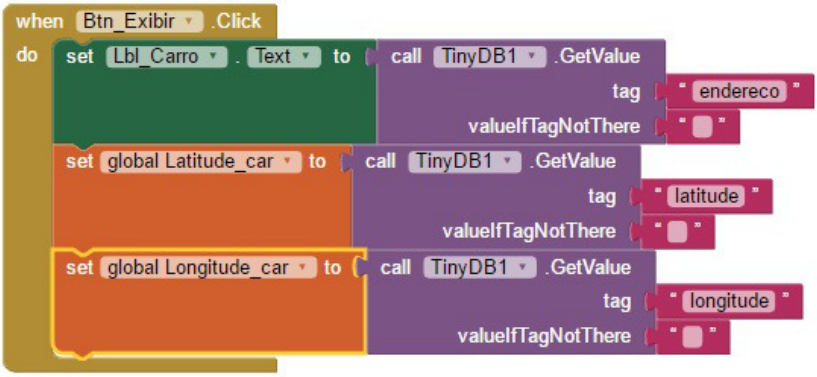

Figura 6.24: Btn_Exibir

6.5 LOCALIZANDO O PONTO DE PARTIDA

Para encontrar o ponto de retorno para o seu veículo, precisamos clicar no botão `Btn_EstouAqui` que identificará as suas coordenadas atuais de localização. Clicando nele, será ativado o geolocalizador `Location_EstouAqui` que então receberá os dados da latitude e longitude para armazenamento nas variáveis `Latitude_atual` e `Longitude_atual` . Futuramente, ele vai traçar e exibir o caminho de chegada, pois precisaremos disso para elaborar a rota entre dois pontos.

Iniciaremos inserindo o bloco do componente `When Btn_EstouAqui.Click` para então habilitar o geolocalizador `Location_EstouAqui`, inserindo o bloco `set Location_EstouAqui to` e encaixando o comando lógico `true` para habilitá-lo. A próxima figura exibe o bloco de ativação do `Location_EstouAqui`.

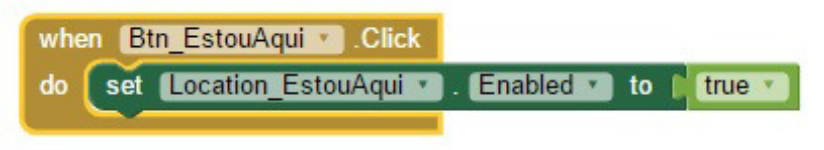

Figura 6.25: Btn_EstouAqui ativando o Location_EstouAqui

Após a sua ativação, automaticamente o componente de localização é executado e os dados são recebidos pelo aplicativo. O bloco que representa esse processo e que deverá ser inserido é o `when Location_EstouAqui.LocationChanched`.

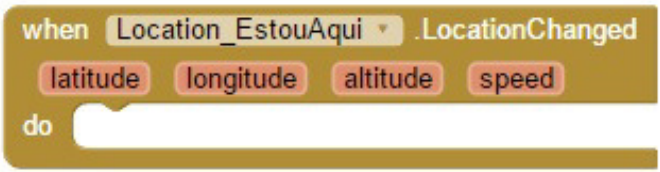

Figura 6.26: Bloco Location_EstouAqui.LocationChanched

Como necessitamos exibir o endereço atual da sua localização indicado pelo bloco `Location_Parado` na `Lbl_Estou`, coloque um componente `Lbl_Estou.Text to`, e nele encaixe o bloco que identificou esse endereço, o `Location.Parado.CurrentAdress`. A figura a seguir mostra o bloco exibindo o endereço atual.

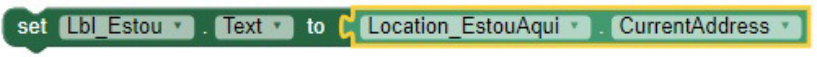

Figura 6.27: Exibindo o endereço atual

O endereço atual exibido serve apenas de informação para o usuário, porém o aplicativo deverá armazenar também a latitude e a longitude nas variáveis `Latitude_atual` e `Longitude_atual`, já que é com esses valores que traçaremos o caminho de volta.

Insira as variáveis `set global Latitude_atual` e `set global Longitude_atual` para receber os valores `get latitude` e `get longitude`, respectivamente. Veja como fica:

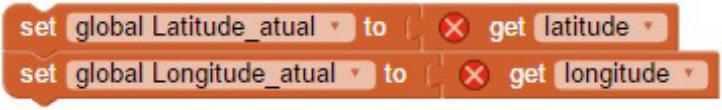

Figura 6.28: Variáveis recebendo os valores da localização atual

Com tudo já preparado, precisamos desabilitar o geolocalizador `Location_EstouAqui`, inserindo um bloco `set Location_EstouAqui to` e encaixando o comando lógico `false`. A próxima figura exibe o bloco de recebimento dos dados `Location_EstouAqui.LocationChanced`.

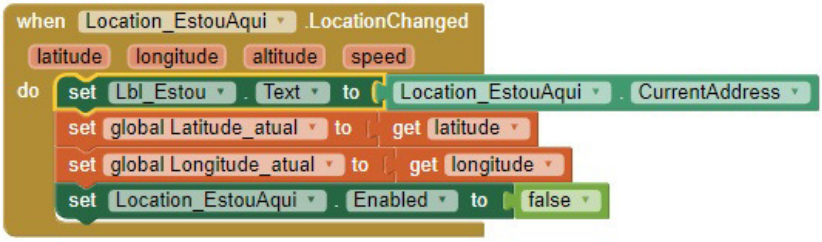
when Location_EstouAqui .LocationChanged
latitude longitude altitude speed
do set Lbl_Estou . Text to Location_EstouAqui . CurrentAddress
set global Latitude_atual to get latitude
set global Longitude_atual to get longitude
set Location_EstouAqui . Enabled to false

Figura 6.29: Bloco Location_EstouAqui.LocationChanched finalizado

6.6 TRAÇANDO A ROTA

Com todos os dados já preparados, precisamos programar o botão que exibirá o caminho de volta até o local do estacionamento do veículo. Aqui usaremos o componente `ActivityStarter1` para chamar a execução da API externa do Google Maps, que exibirá o caminho de retorno. Para isso, basta informar a URL da API, as coordenadas atuais e as iniciais, e depois inicializar o `ActivityStarter` para exibir o mapa.

Insira primeiramente o botão que executará todos os demais comandos, o `When Btn_Chegar.Click` . Dentro dele, necessitamos de um bloco `set ActivityStarter1.DataUri to` para informarmos o endereço da API do Google Maps e os pontos de partida e chegada.

Como temos várias informações para passar à API, é preciso colocar um bloco de texto `join` para juntar todos os dados. Para isso, acesse a guia `Blocks` e, na seção `Built-in` , selecione a guia dos blocos de `Text` para expandir suas opções e encontrar o bloco de texto `join` . Note que ele já vem configurado para receber duas informações.

No primeiro espaço da `join` , insira um bloco de texto e, dentro dele, informe o endereço `http://maps.google.com/maps?saddr=` . No segundo espaço, insira a variável `get global Latitude_atual` .

Configure sua `join` para receber mais seis opções de entradas de textos, pois necessitamos informar as coordenadas iniciais e finais, separadas por uma vírgula, e o comando que indica o ponto de destino: `daddr` .

A latitude e a longitude sempre serão separadas por vírgula (,). Adicione um bloco de texto e, dentro dele, insira essa pontuação.Na próxima entrada de texto da `join` , acrescente a variável `get global Longitude_atual` .

Já temos o ponto de partida, precisamos configurar agora o ponto de chegada, informando as coordenadas iniciais de onde estacionamos nosso veículo. Para dizer à API que informaremos o ponto de chegada, devemos acrescentar um bloco de texto com a informação `&daddr=` e, na sequência da entrada de textos, encaixar em cada opção as variáveis do destino `get global Latitude_car` e `get global Longitude_car` , não se esquecendo de colocar uma vírgula entre os blocos das variáveis.

Atenção

- `saddr=` : serve para definir o ponto de partida para as pesquisas de rotas. Esse ponto pode ser uma latitude, uma longitude ou um endereço formatado para consulta.
- `daddr=` : define o ponto de chegada para as pesquisas de rotas. Tem o mesmo formato e comportamento de `saddr=` .

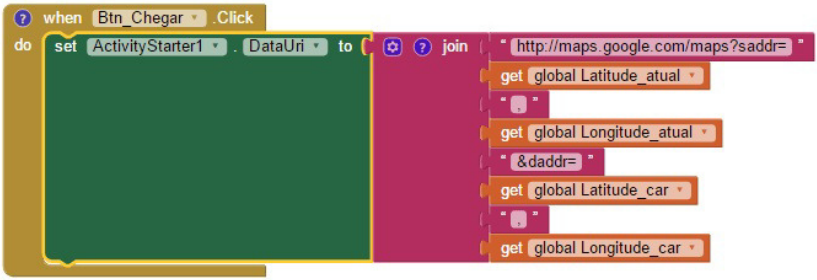

Figura 6.30: Btn_Chegar com o ActivityStarter configurado

Já está tudo pronto para a exibição da API, basta agora solicitar a execução. Coloque o bloco que realiza a chamada da API, o `call ActivityStarter1.StartActivity` . A figura a seguir exibe o botão `Chegar` com todos os códigos necessários.

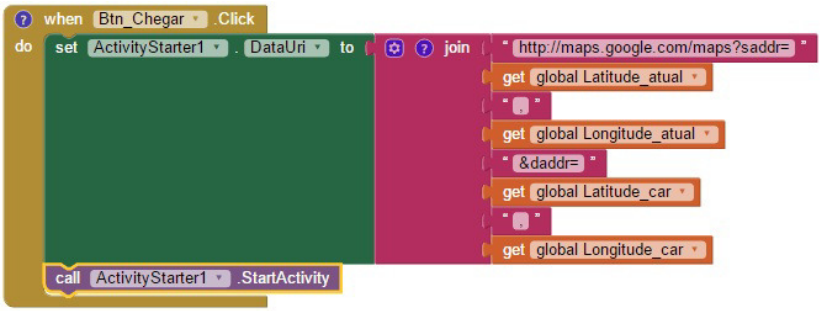

Figura 6.31: Btn_Chegar com todos os códigos

6.7 TESTANDO SEU APLICATIVO

Para testar esse aplicativo, será preciso instalá-lo em seu dispositivo. Essa operação é necessária devido à utilização do geolocalizador que está presente em seu celular. Vale lembrar que, para a exibição do mapa com a rota traçada, você precisa ter acesso à internet.

Após a instalação, abra o aplicativo e clique no primeiro botão à esquerda. Quando o localizador encontrar o endereço, este será exibido na tela e, na sequência, solicitará a sua decisão para salvar ou não o endereço do estacionamento.

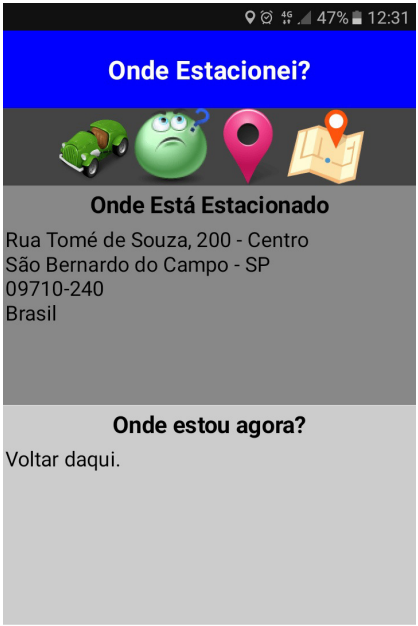

Figura 6.32: Localizando o endereço do estacionamento

O segundo botão da direita para a esquerda exibe o local atual onde você se encontra. Serve para traçar a rota de volta até o local de estacionamento. A figura a seguir exibe um exemplo de local para iniciar o retorno.

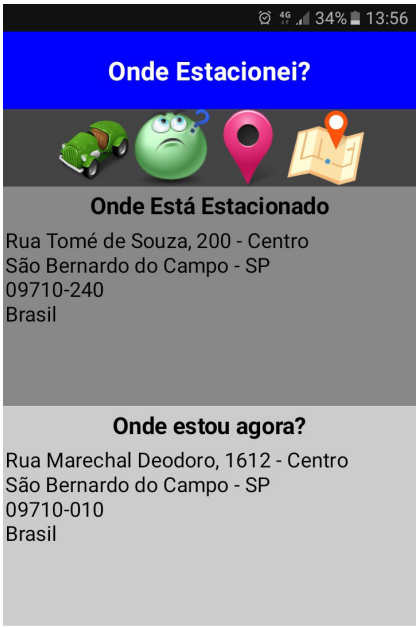

Figura 6.33: Exibindo um local para iniciar o retorno

Quando clicar no primeiro botão à direita, será exibido um mapa com a rota para você chegar até seu veículo. A próxima imagem mostra o mapa com a rota de volta ao ponto de partida.

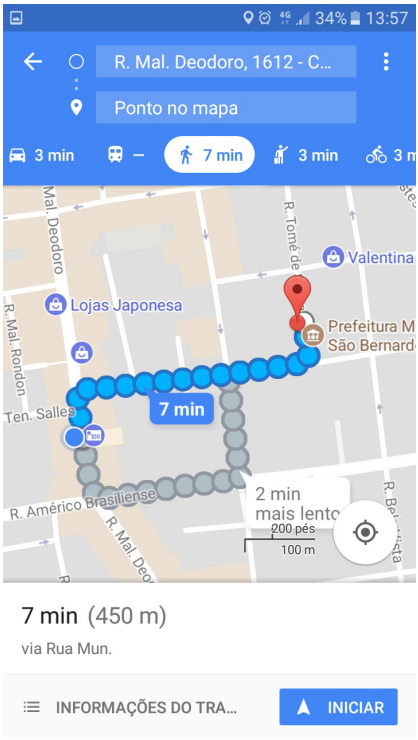

Figura 6.34: Exibindo o mapa com a rota

6.8 RESUMINDO

Neste capítulo, aprendemos a trabalhar com o geolocalizador do seu dispositivo, salvar os valores em um `TinyDB` e utilizar a API do Google Maps para traçar uma rota.

No próximo capítulo, iniciaremos o projeto que fará o nosso aplicativo acessar um banco de dados em MySQL. Veremos primeiramente a configuração do **ambiente de desenvolvimento web** *Apache* em seu computador. Assim, o App Inventor se conectará ao gerenciador de banco de dados *PhpMyAdmin* contido

nesse servidor.

Capítulo 7

CONFIGURAÇÃO DO SERVIDOR WEB

A partir de agora, iniciaremos os estudos para conectar um aplicativo com um banco de dados. Vamos dividir os passos em capítulos para demonstrá-lo de uma forma mais didática.

Utilizaremos o banco de dados MySQL. A princípio, faremos a conexão com um banco de dados local para realizarmos os testes, e futuramente mostraremos como deixá-lo online.

Nesse primeiro passo, vamos instalar um servidor web local que possua o gerenciador de banco de dados em MySQL, pois ele tem como foco sistemas online. Logo após, faremos o emulador do App Inventor reconhecê-lo para realizar a conexão.

7.1 CONFIGURANDO O SERVIDOR WEB LOCAL

Antes de realizarmos as tarefas de conexão do App Inventor com o banco de dados MySQL, vamos conhecer os passos e as maneiras de deixar nosso ambiente todo configurado e pronto para uso. Precisamos ter instalado em nosso computador um servidor web com o gerenciador de banco de dados MySQL.

Você poderá utilizar algumas opções existentes, como o *WampServer* ou *VertrigoServ*, mas vou apresentar aqui uma excelente alternativa que pode ser executada também em uma mídia removível como pendrive, cartão de memória ou HD externo. Trata-se do aplicativo **USBWebserver**, um software gratuito.

As vantagens da utilização do USBWebserver são:

- Não precisa de instalação! É só descompactar e usar.
- É mais leve para executar, pois o arquivo é extremamente pequeno.
- Tem uma utilização mínima de memória e não exige grande capacidade de processamento de seu computador.
- É muito mais prático de operar.
- Não precisa estar obrigatoriamente em seu *pendrive*, podendo ser descompactado e utilizado diretamente em um PC.

Se ele estiver salvo em seu pendrive, qualquer computador que possua o sistema operacional Windows estará pronto para executar o servidor web local. Não precisamos de novas instalações, como o WampServer e o VertrigoServ exigiriam. Basta descompactar e usar. Por estar em uma mídia removível, você poderá utilizá-lo em qualquer computador sem a necessidade de realizar configurações ou instalações nesse novo ambiente.

Comecemos pelo download do USBWebserver pelo link http://www.nelfabbri.com/appinventor/usbwebserver.zip. Após

baixar, descompacte-o em seu pendrive ou em uma pasta do seu computador. Depois, abra o local onde foi gravado para ver a estrutura das pastas geradas. A figura a seguir exibe como seus arquivos deverão estar:

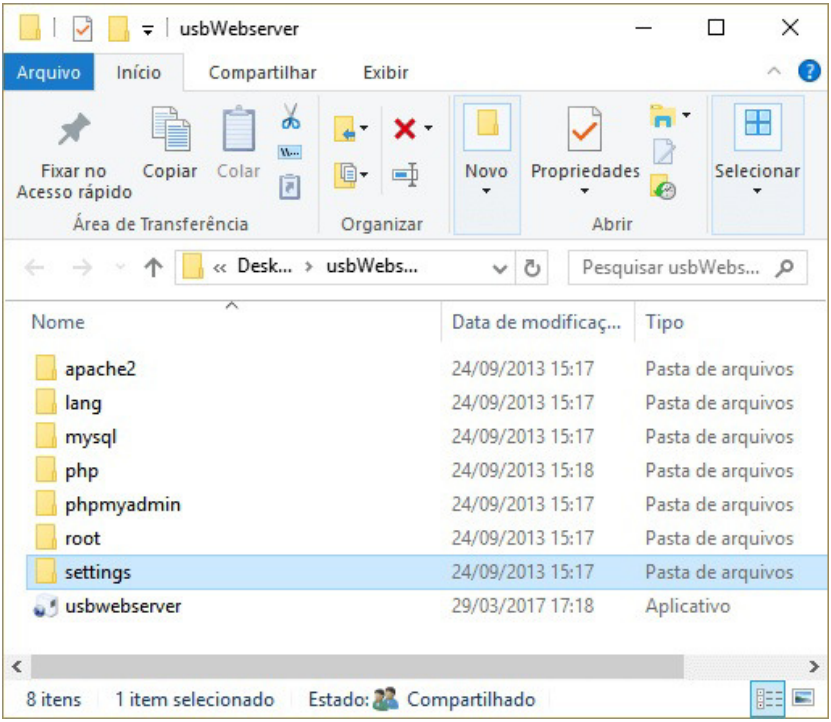

Figura 7.1: Arquivos do USBWebserver descompactados

Agora, basta dar dois cliques sobre o aplicativo de nome USBWebserver para executá-lo. A figura a seguir mostra como ficará visível em sua tela o aplicativo funcionando.

Figura 7.2: Tela do USBWebserver em execução

Note que, ao lado das opções Apache e MySQL, temos um ícone verde, indicando que os serviços do Apache e MySQL estão funcionando perfeitamente em seu computador:

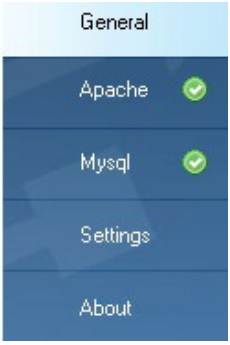

Figura 7.3: Exibindo os serviços ativos

Caso algum desses ícones fique na cor vermelha, isso indicará que não conseguimos deixar o servidor web local ativo. Apenas neste caso será necessário realizarmos algumas pequenas configurações referentes às **portas de comunicação** que o USBWebserver precisa usar.

Portas de comunicação

Os dados entram e saem de nossas máquinas pela internet por meio dessas portas de comunicação. Se uma porta está fechada, a comunicação não ocorrerá, logo, o servidor web local não funcionará, necessitando que alteremos os números da porta utilizada.

Clique na opção `Settings` para acertar as configurações das portas do Apache e/ou MySQL. Veja na figura a seguir onde você poderá alterar as informações.

Figura 7.4: Alteração das portas do servidor

Faça essas alterações somente se os ícones ao lado das opções Apache e/ou MySQL não estiverem na cor verde. Normalmente a porta de número 80 não necessita ser alterada, mas, caso precise, utilize uma das portas alternativas: 81, 82, 90, 591, 8080, 8686 ou 9090.

Após a alteração, clique no botão `Save` , reinicie seu aplicativo USBWebserver e veja se os ícones estão na cor verde. Caso não estejam, refaça esses passos alterando apenas a porta necessária.

Caso ainda haja problemas, certifique-se de que você tem espaço suficiente no seu pendrive. Permanecendo o problema, realize a cópia de toda as pastas do aplicativo USBWebserver para o seu computador e refaça as alterações das portas de comunicação, conforme demonstrado.

Como já vimos, a escolha pela execução do USBWebserver através de um pendrive foi apenas para facilitar o seu dia a dia. Assim, você poderá utilizar outros computadores para seu estudo e continuar com todas as informações referentes ao servidor web local configuradas, não precisando baixar ou instalar nada nessa nova máquina.

Estando o servidor web com os símbolos verdes, clique na aba `General` e no botão que indica `localhost` , para que se abra a página inicial, indicando que o servidor está em perfeita execução em seu computador. Verifique na próxima figura como deverá aparecer no seu navegador padrão a página inicial do USBWebserver.

Figura 7.5: A página localhost que será exibida

7.2 LOCALIZANDO O NÚMERO IP DE SEU COMPUTADOR

Necessitamos identificar o número IP do seu computador, para que o App Inventor reconheça o servidor web local e possa se conectar para acessar o banco de dados que será criado no próximo capítulo. Para identificar o seu IP interno, precisamos abrir a janela **executar** do Windows e localizar o número do IP.

Pressione simultaneamente as teclas `Windows + R` para abrir a janela `Executar` de seu Windows. Digite no espaço em frente à opção **Abrir** o comando `cmd`, e clique no botão `OK` para prosseguir.

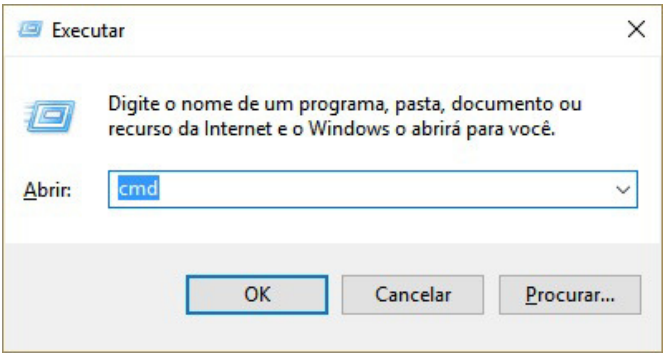

Figura 7.6: Tela Executar do Windows

Será exibida uma janela do **prompt de comando**:

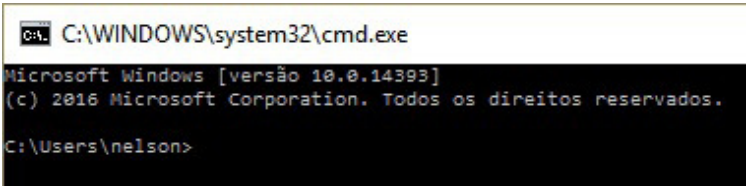

Figura 7.7: Prompt de comando

Nela, digite apenas o comando `ipconfig` e pressione a tecla `Enter`. Note na próxima figura que você encontrará o número de IP interno de seu computador na linha onde consta o endereço **IPv4**. Anote-o, pois precisaremos dele mais adiante.

Todas as vezes que você utilizar uma conexão com a internet, uma rede diferente ou outro computador, você precisará identificar novamente o número de IP.

Vale lembrar que estamos trabalhando ainda localmente. Quando usarmos um banco de dados online, não haverá mais essa necessidade.

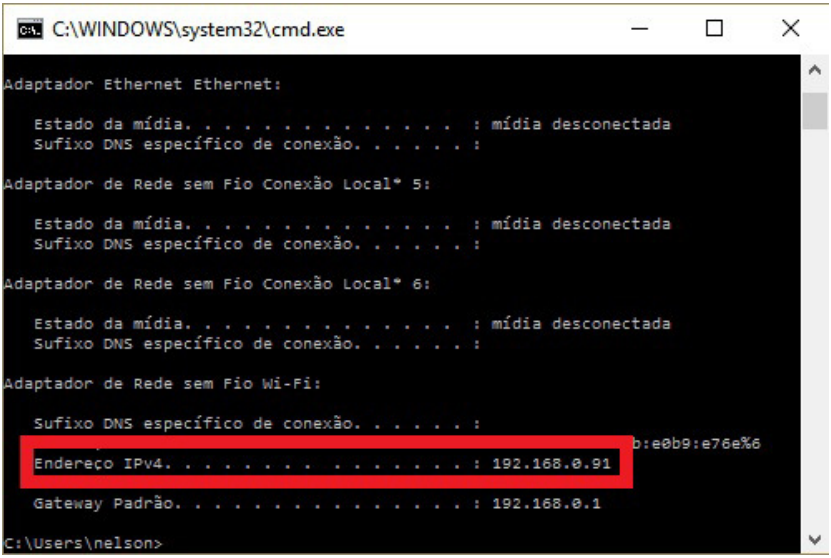
C:\WINDOWS\system32\cmd.exe
Adaptador Ethernet Ethernet:
Estado da mídia. . . . . : mídia desconectada
Sufixo DNS específico de conexão. . . . . :
Adaptador de Rede sem Fio Conexão Local* 5:
Estado da mídia. . . . . : mídia desconectada
Sufixo DNS específico de conexão. . . . . :
Adaptador de Rede sem Fio Conexão Local* 6:
Estado da mídia. . . . . : mídia desconectada
Sufixo DNS específico de conexão. . . . . :
Adaptador de Rede sem Fio Wi-Fi:
Sufixo DNS específico de conexão. . . . . :
b:e0b9:e76e%6
Endereço IPv4. . . . . : 192.168.0.91
Gateway Padrão. . . . . : 192.168.0.1
C:\Users\nelson>

Figura 7.8: Exibindo o endereço do IPv4

Após anotar os números do IP, você poderá fechar essa janela digitando o comando `exit` e, na sequência, pressionando `Enter` ; ou simplesmente clicando no botão fechar da janela ativa do Windows.

7.3 RECONHECENDO O SERVIDOR PELO EMULADOR

Agora que já temos o servidor web instalado e rodando em nosso computador, e também identificamos o número IP, vamos abrir o site do App Inventor para que ele faça a identificação do seu servidor.

Crie um novo projeto e dê o nome de `CONTROLE_DE_CORRIDAS`. Não use espaços entre as palavras, pois não é permitido pelo App Inventor; no local dos espaços, utilize o caractere *underline* (_). Nesse momento, não se preocupe com o objetivo do aplicativo que vamos desenvolver, porque explicaremos todos os detalhes nos próximos capítulos. Por hora, podemos dizer que ele realizará o agendamento de informações de atividades esportivas.

Espere a tela principal do App Inventor ser carregada e, mesmo sem nenhuma informação em nosso aplicativo, vamos colocá-lo para realizar o teste de conexão com o servidor web local. Clique no menu suspenso `CONNECT`, selecione `EMULATOR`, e aguarde o emulador carregar.

Você vai visualizar uma tela vazia, conforme a figura a seguir. Precisaremos abrir o navegador da internet do emulador para inserir o número do IP e verificar se a conexão com o servidor web foi realizada. Clique no ícone (voltar) em destaque na imagem para acessar a tela principal do dispositivo.

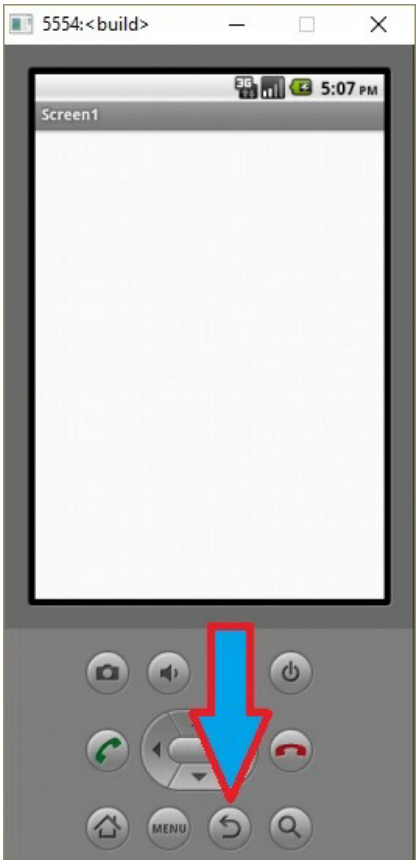

Figura 7.9: Retornando para a tela principal

Vamos abrir o navegador no emulador para realizarmos o teste de conexão, clicando no ícone do navegador:

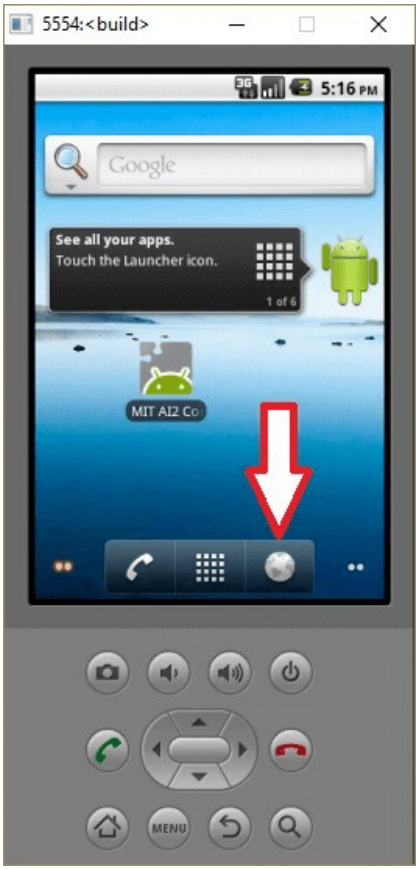

Figura 7.10: Abrindo o navegador de internet

Com o navegador aberto, digite na linha de endereço de navegação o seu número de IP e clique no botão de navegar. A página inicial do seu servidor web deverá ser exibida, conforme demonstra a próxima figura.

O leitor deverá digitar o endereço no formato: **http://seunumeroIP**. Caso não apareça a tela inicial do servidor web, insira o número da porta de comunicação usada após o seu

número de **IP**, com o símbolo de : (dois pontos). Nesse caso, o formato final do endereço deverá ser: **http://seunumeroIP:8080**.

Figura 7.11: Abrindo o navegador de internet

Se no seu emulador aparecer a tela inicial do USBWebserver conforme a imagem anterior, isso indicará que o leitor obteve sucesso na tarefa de habilitar o servidor web local em seu dispositivo e estará apto para dar continuidade a nossos estudos. Mas caso ocorra outra tela, solicito que faça a releitura deste

capítulo, pois sem isso não conseguiremos prosseguir.

7.4 RESUMINDO

Aprendemos como baixar o servidor web local **USBWebserver**, colocando-o em funcionamento para futura utilização. Identificamos o número de IP do computador e realizamos o reconhecimento do App Inventor com aquele servidor através desse número.

Essas são as configurações necessárias para que nosso aplicativo se conecte a um banco de dados. No próximo capítulo, veremos como criar um banco de dados, com tabelas e campos no gerenciador do MySQL, para usarmos no projeto `CONTROLE_DE_CORRIDAS`.

Capítulo 8

CRIAÇÃO DO BANCO DE DADOS COM O MYSQL

Após a configuração do ambiente do App Inventor para o reconhecimento do servidor web local, criaremos um banco de dados para podermos acessar com o aplicativo que vamos desenvolver. Apresentaremos o *phpMyAdmin*, gerenciador de banco de dados do MySQL e parte integrante do *USBWebserver*. Nele, criaremos a estrutura do banco de dados do projeto de cadastro de corridas.

O phpMyAdmin é um aplicativo livre e de código aberto para o gerenciamento das informações de um banco de dados em MySQL. Com ele, é possível: criar e remover um banco de dados; criar, remover e alterar tabelas; inserir, remover e editar os campos dessa tabela. É o phpMyAdmin que armazenará todas as informações digitadas através do app que vamos desenvolver.

8.1 O APLICATIVO

Como todo aplicativo surge de uma necessidade, este também veio para me auxiliar. Além de profissional de TI, sou praticante de corrida de rua e já realizei algumas maratonas (42.195 metros) e tantas outras provas de distâncias diferentes. O agendamento

dessas corridas sempre acaba sendo uma dificuldade, pois me inscrevo em provas durante o ano, com muita antecedência, e acabo esquecendo quando ocorrerão — já cheguei a me inscrever em duas corridas no mesmo dia!

Pensando nesse problema, vamos criar uma agenda de corridas. Nela poderemos cadastrar, alterar, consultar e excluir nome, local, data, distância a percorrer e horário da largada de uma prova. Espero assim suprir essa dificuldade e lhe apresentar um aplicativo interessante que realize as movimentações possíveis com um banco de dados. Deixo claro ao leitor que você poderá criar outros aplicativos futuramente com base no que verá nos capítulos a seguir.

Apresentada a ideia do aplicativo, vamos preparar a criação da estrutura do banco de dados.

8.2 CONHECENDO O PHPMYADMIN

Agora que já instalamos o servidor web (USBWebserver), descobrimos nosso número IP e já testamos o emulador do App Inventor para acessar o *localhost*, continuaremos o desenvolvimento, criando o banco de dados do aplicativo.

Precisamos abrir o gerenciador de banco de dados, o phpMyAdmin. Volte à janela do USBWebserver, exibida na figura a seguir, e clique na opção `PHPMyAdmin` para entrarmos no ambiente de criação do banco.

Figura 8.1: Janela do USBWebserver

Após o clique, será aberta em seu navegador a página de entrada para a criação do banco de dados:

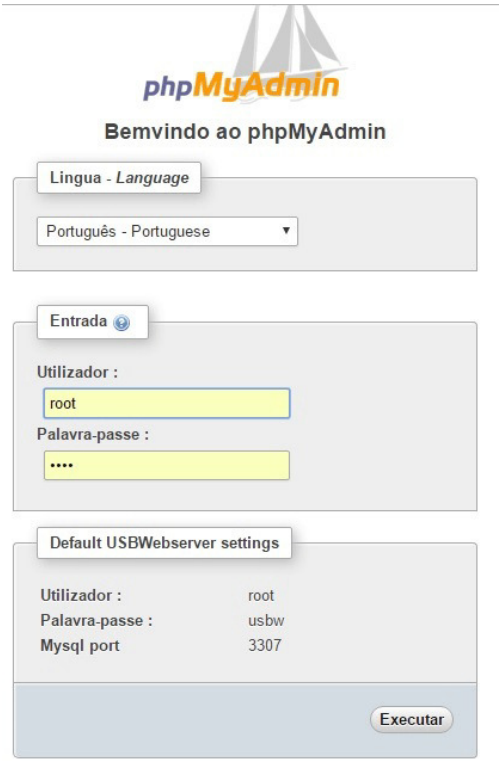

Figura 8.2: Tela principal do phpMyAdmin

Para acessar o ambiente do gerenciador de banco de dados, é necessário informar o nome do `Utilizador` e a `Palavra-passe`. Observe na imagem anterior que essas informações estão disponíveis na parte inferior da tela, na seção `Default USBWebserver settings`.

Digite na opção `Utilizador` o nome padrão do usuário (`root`) e, na opção da `Palavra-passe`, escreva `usbw`. Solicito uma especial atenção do leitor nesse momento, pois todas as letras digitadas deverão estar em minúsculas, já que o `phpMyAdmin` é

case-sensitive.

> **CASE-SENSITIVE** significa que caracteres em caixa alta (maiúsculas) e em caixa baixa (minúsculas) são tratados de modo diferente. Por exemplo, as palavras **root** e **Root** são consideradas diferentes.

Digite essas informações nas caixas específicas e clique no botão `Executar` para acessar o ambiente.

8.3 DEFININDO O BANCO DE DADOS DO APLICATIVO

Se você digitou tudo corretamente, será exibida a tela da figura a seguir, solicitando a criação de um novo banco de dados. Digite `CORRIDA`, que será o seu nome, e depois clique no botão `Criar` para efetivar sua solicitação.

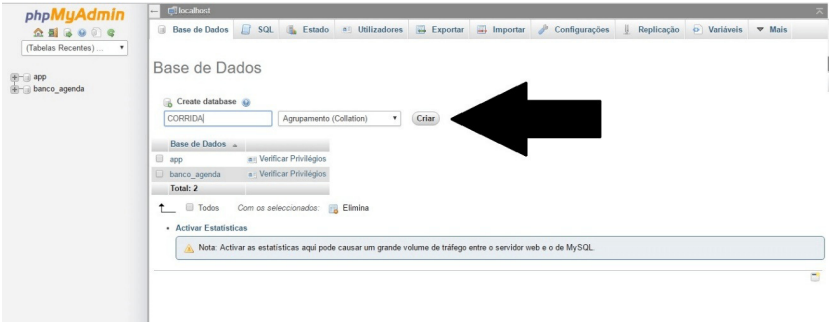

Figura 8.3: Criando o banco de dados

Na sequência, serão solicitados o nome da tabela e a quantidade de campos que ela terá. Vamos definir aqui a **estrutura de dados da tabela**. A próxima figura demonstra a tela nesse momento.

> Uma ESTRUTURA DE DADOS é um modo de armazenamento e organização dos dados em um computador, possibilitando que eles sejam usados futuramente. É nessa estrutura que vamos armazenar todos os dados das corridas cadastradas através do app desenvolvido nos próximos capítulos.

Para definir o nome da tabela, digite no campo `Nome` a palavra `AGENDA` e, em `Numbers of columns`, a quantidade de colunas dessa tabela. Teremos os seguintes campos: `Codigo`, `Nome`, `Local`, `Distancia`, `Data` e `Largada`, logo, um total de 6. Clique no botão `Executar` para realizar a sua solicitação.

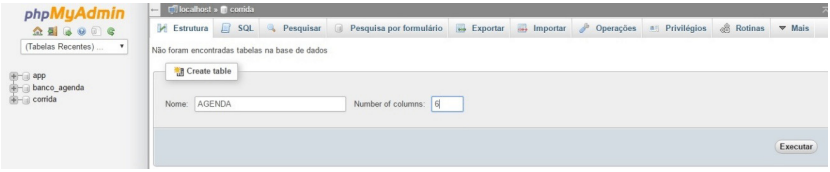

Figura 8.4: Criando a tabela agenda

Surgirá a tela onde devemos digitar as informações de cada campo de nossa tabela do banco de dados:

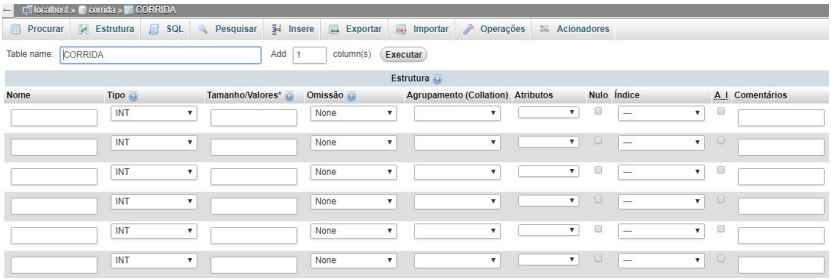

Figura 8.5: Tela para criação dos campos

Veja a seguir os campos da tabela `AGENDA` que devemos digitar nos espaços disponíveis na tela do phpMyAdmin.

Nome	Tipo	Tamanho	Índice	A_I	Comentários
Codigo	INT		Primary	Selecionar	
Nome	Varchar	30			Nome da Corrida
Local	Varchar	30			Local da Corrida
Distancia	Varchar	10			Distância a percorrer
Data	Date				Data do Evento
Largada	Time				Hora da largada

Os nomes do título da tabela representam:

- **Nome**: o nome que identifica o campo da tabela.
- **Tipo**: o tipo da informação do campo, podendo ser texto, número, data, hora ou outra informação necessária.
- **Tamanho**: o comprimento do campo — quantos caracteres

ele poderá armazenar.

- **Índice**: sua função é acelerar o tempo de acesso às informações gravadas na tabela.
- **A_I**: este é o campo autoincremento. A cada novo registro, automaticamente será criado um número sequencial para a identificação do código.
- **Comentários**: informação que aparecerá apenas no phpMyAdmin para nos auxiliar a identificar cada campo criado.

As colunas que estiverem em branco não necessitam de informações, bem como as outras opções que estão na tela da criação da tabela.

Toda tabela necessita de um campo definido como `Índice`, já que sua função é acelerar o tempo de acesso às informações gravadas na tabela. Criamos o campo `Codigo` para essa finalidade. Para isso, na coluna do `Índice`, devemos selecionar a opção `PRIMARY`. A coluna `A_I` indicará que o código terá sua numeração incrementada automaticamente, ou seja, a cada novo registro que for cadastrado, será gerado um novo número para o campo `Codigo`.

Na segunda linha para digitação, criaremos o campo `Nome`, pois ele armazenará o nome da corrida a ser cadastrada. Na coluna `Tipo`, definimos o campo como `Varchar`, pois ele vai guardar um nome, e esse tipo de dados comporta qualquer caractere. Já na coluna `Tamanho`, temos de informar a quantidade de caracteres que esse campo poderá armazenar, então definimos como `30`.

Para os campos `Local` e `Distancia`, que armazenam o lugar onde acontecerá a prova e a distância a ser percorrida, definimos também os tipos para `Varchar` e os tamanhos para `30` e `10`, respectivamente.

No campo que arquivará a `Data` do evento, definimos seu tipo como `Date`, pois esse é o melhor tipo de dado para guardar tal informação. O último campo `Largada` armazenará o horário da partida da corrida, então deixamos a coluna `Tipo` como `Time`.

Solicito uma atenção especial para este momento da criação do banco de dados, devido às letras maiúsculas e minúsculas. Digite os nomes dos campos exatamente como estão, evitando também os caracteres especiais, como os acentos nas palavras. Caso esteja utilizando algum caractere diferente dos apresentados, você certamente encontrará um erro futuro durante a execução dos comandos que veremos adiante.

Veja na figura a seguir como deverá estar sua tabela após a digitação das informações:

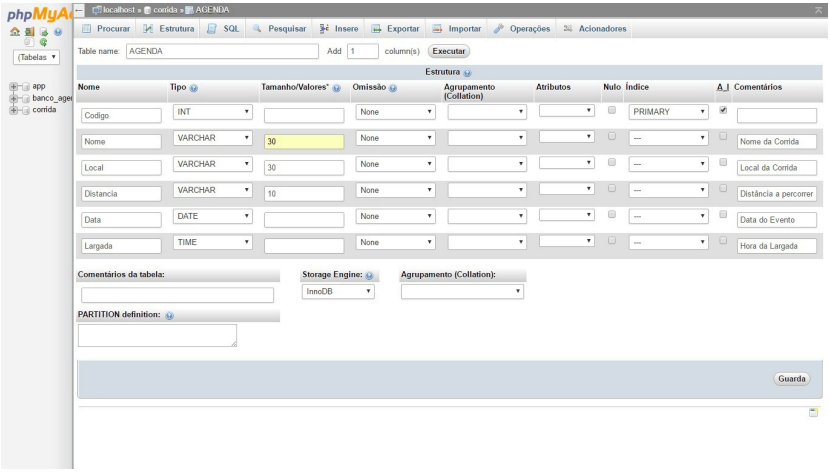

Figura 8.6: Dados da tabela agenda

Para salvar as informações do banco de dados, clique no botão `Guarda` . Veja a seguir como ficará sua tabela no menu lateral esquerdo, após clicar no botão. Caso seja necessário, expanda os nós para a exibição dos campos.

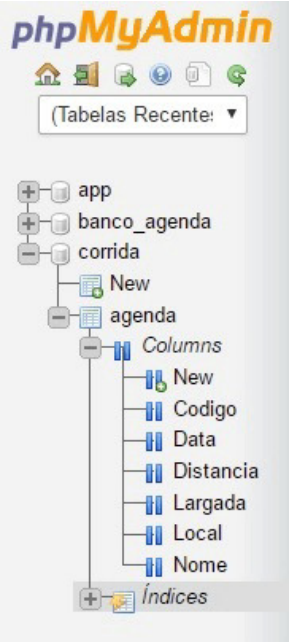

Figura 8.7: Visualização dos campos da tabela criada

8.4 RESUMINDO

O fechamento da janela de navegação do phpMyAdmin é opcional para a continuidade dos trabalhos do próximo capítulo. Acabamos de criar o banco de dados `corrida`, com uma tabela chamada `agenda` e seis campos. Aqui concluímos mais uma etapa do processo para a criação do app com acesso ao banco de dados.

No próximo capítulo, criaremos o layout do aplicativo para controle de corridas e já começaremos a programar os blocos para inserir registros no banco.

CAPÍTULO 9

LAYOUT DO APP

Após configurar o servidor web local, identificar o número do IP e criar o banco de dados, vamos agora desenvolver o *layout* do aplicativo de controle de corridas.

Iniciaremos desenvolvendo o *layout* de nosso app com as caixas de textos para a digitação das informações referentes às corridas que serão adicionadas e outros componentes para nos auxiliar. Neste capítulo, criaremos apenas os blocos de programação do botão que realiza o cadastro da informação no banco de dados, ficando os demais controles (consulta, exclusão e alteração) para os próximos.

9.1 DEFININDO OS COMPONENTES DE TELA

Vamos conhecer a tela que produziremos após inserir os componentes e realizar as configurações das propriedades necessárias:

Figura 9.1: Tela do aplicativo de corridas

Retorne ao ambiente de desenvolvimento do App Inventor para configurar e criar a tela do aplicativo `CONTROLE_DE_CORRIDAS`.

Sem inserir nenhum componente na `Screen1`, altere na guia `Properties` o nome do aplicativo. Esse nome será exibido quando ele for instalado em seu celular ou tablet. A propriedade `App Name` deverá ser alterada para **Minhas Corridas**. Não se preocupe com o fato de o nome do projeto ser diferente do `App Name`, pois um é o nome que define o projeto e o outro é o que será exibido quando o aplicativo for instalado.

Para que o dispositivo não altere a orientação da página, devemos alterar a propriedade `ScreenOrientation` para `Portrait`. Como vimos anteriormente, não desejamos que a barra de título do aplicativo fique visível, pois criaremos a nossa própria. Para isso, desmarque a propriedade `TitleVisible`. Ao desmarcá-la, você não perceberá nenhuma mudança durante a criação do design, pois ela só funcionará durante a emulação.

Para criar uma barra de títulos no nosso aplicativo, devemos inserir na `Screen1` um componente `HorizontalArrangement` da guia `Layout`. Nele precisamos alterar as propriedades de alinhamento vertical e horizontal para centralizar as informações que serão inseridas. Altere `AlignVertical` e `AlignHorizontal` para `Center`. Na propriedade `BackgroundColor`, escolha `Blue` para alterar a cor de fundo do componente para azul.

Precisamos definir um tamanho fixo de altura da barra de título. Na propriedade `Height`, altere o valor para `40` pixels. Alteraremos também a largura para que ocupe o espaço total da tela. Vá à propriedade `Width` e selecione a opção `Fill Parent`. Para concluir as modificações do `HorizontalArrangement`, troque seu nome para `Horizontal_Titulo`.

Já temos a barra de título pronta, basta exibir uma mensagem em seu interior. Insira um componente `Label` da guia `User Interface` dentro do `HorizontalArrangement` que se encontra na área `Viewer`. Com a `Label` selecionada, ative a propriedade `FontBold` para o efeito de negrito.

Vamos aumentar o tamanho da fonte também. Na propriedade `FontSize`, altere para `20`. Adicione o título do aplicativo alterando a propriedade `Text` para `Minhas Corridas`. Para um

maior destaque, mudaremos a cor do texto para branco, assim, na propriedade `TextColor`, altere para `White`. Renomeie também a `Label` para `Lbl_Titulo`. Nesse momento, você deverá verificar a tela conforme a figura a seguir:

Figura 9.2: Barra de título

Para dar um espaçamento entre a barra de título e os demais componentes, colocaremos um `HorizontalArrangement` logo abaixo dela, que já está configurado em sua tela. Realize as alterações apenas das propriedades: `Height` para `45` pixels, e `Width` para `Fill Parent`. Veja como ficará sua tela:

Figura 9.3: Espaçamento inserido

Para um layout mais organizado, cada informação da corrida que digitarmos será organizada dentro de um `HorizontalArrangement`. Então, insira mais um logo abaixo da área de espaçamento e realize as seguintes modificações em suas propriedades: `AlignVertical` para `Center`, `AlignHorizontal` para `Center`, `Height` para `45` pixels, e

`Width` para `Fill Parent`.

Precisamos de uma `Label` para identificar a informação que será digitada. Insira-a dentro do `HorizontalArrangement` e, para melhor destacar a informação, altere algumas das suas propriedades, tais como: negrito (`FontBold`), tamanho (`FontSize` altere para `16`) e cor da fonte (`TextColor` para `White`), além do texto de exibição. Em `Text`, digite a palavra **Corrida**. Altere também o nome da `Label` para `Lbl_Corrida`.

Agora vamos inserir mais uma `Label` após a `Lbl_Corrida`, que terá como função apenas dar um espaçamento entre essa última e a `TextBox`. Mas atenção, esse recurso não ficará visível na tela de design, somente terá efeito na emulação do app. Na propriedade `Text`, apague o que estiver escrito e insira 4 espaços em branco. Altere o nome dessa `Label` para `Lbl_espaco1`.

Para o usuário poder digitar o nome da corrida, precisamos inserir um componente `TextBox`. Em `FontSize`, altere o tamanho da fonte para `16`. A propriedade `Hint`, que exibe um texto de informação ao usuário, deverá ser apagada para termos um layout mais limpo. Renomeie sua `TextBox` para `Txt_Corrida`. Veja como ficará momentaneamente a tela:

Figura 9.4: Digitação do nome da corrida

Para os próximos dois campos a serem digitados, **Local** e **Distância** da corrida, devemos fazer da mesma maneira. Na tabela a seguir temos os componentes a serem inseridos, bem como suas propriedades.

Componente	Pallete	Propriedade	Alterar valor para
HorizontalArrangement	Layout	AlignVertical	Center
		AlignHorizontal	Center
		Height	45 pixels
		Width	Fill Parent
Label	User Interface	FontBold	Marcar
		FontSize	16
		Text	Local
		TextColor	White
		Rename	Lbl_Local
Label	User Interface	Text	4 espaços
		Rename	Lbl_espaco2
TextBox	User Interface	FontSize	16
		Hint	Apagar
		Rename	Txt_Local
HorizontalArrangement	Layout	AlignVertical	Center
		AlignHorizontal	Center
		Height	45 pixels
		Width	Fill Parent

Componente	Pallete	Propriedade	Alterar valor para
Label	User Interface	FontBold	Marcar
		FontSize	16
		Text	Distância
		TextColor	White
		Rename	Lbl_Distancia
Label	User Interface	Text	4 espaços
		Rename	Lbl_espaco3
TextBox	User Interface	FontSize	16
		Hint	Apagar
		Rename	Txt_Distancia

A figura exibe como a tela deverá estar após a inserção e configuração dos componentes:

Figura 9.5: Tela com os três campos

Os dois campos que ainda estão faltando, **data** e **horário** da corrida, seguem a mesma lógica vista anteriormente: com um

`HorizontalArrangement` , duas `Labels` e uma `TextBox` em seu interior. Além deles, vamos conhecer outros dois componentes da guia `User Interface` .

O primeiro é o `DatePicker` . Ele exibirá um calendário para a seleção da data. O segundo é o `TimePicker` , que exibirá um relógio digital para a escolha do horário de largada da corrida.

Acrescente na `Screen1` os componentes da tabela a seguir e realize as alterações necessárias.

Componente	Pallete	Propriedade	Alterar valor para
HorizontalArrangement	Layout	AlignVertical	Center
		AlignHorizontal	Center
		Height	45 pixels
		Width	Fill Parent
Label	User Interface	FontBold	Marcar
		FontSize	16
		Text	Data
		TextColor	White
		Rename	Lbl_Data
Label	User Interface	Text	4 espaços
		Rename	Lbl_espaco4
TextBox	User Interface	FontSize	16
		Hint	Apagar
		Rename	Txt_Data

Componente	Pallete	Propriedade	Alterar valor para
DatePicker	User Interface	Height	35 pixels
		Width	35 pixels
		Text	Apagar o texto
		image	Realizar o upload do arquivo calendar.png
HorizontalArrangement	Layout	AlignVertical	Center
		AlignHorizontal	Center
		Height	45 pixels
		Width	Fill Parent
Label	User Interface	FontBold	Marcar
		FontSize	16
		Text	Largada
		TextColor	White
		Rename	Lbl_Largada
Label	User Interface	Text	4 espaços
		Rename	Lbl_espaco5
TextBox	User Interface	FontSize	16
		Hint	Apagar
		Rename	Txt_Horario
TimePicker	User Interface	Height	35 pixels
		Width	35 pixels
		Text	Apagar o texto
		image	Realizar o upload do

			arquivo timer.png

Não se esqueça de baixar as imagens usadas no projeto em http://nelfabbri.com/appinventor/appinventor.zip.

A figura a seguir exibe o atual estágio em que sua tela deve estar:

Figura 9.6: Todos os campos posicionados

Precisamos criar uma área reservada aos botões de controle que realizarão as operações de incluir, consultar, excluir e alterar os registros do banco de dados. Para isso, inseriremos mais um componente `HorizontalArrangement` . Vale lembrar que, neste capítulo, falaremos apenas do botão que vai **salvar** os dados digitados.

Componente	Pallete	Propriedade	Alterar valor para
HorizontalArrangement	Layout	AlignVertical	Center
		AlignHorizontal	Center
		Height	45 pixels
		Width	Fill Parent
Button	User Interface	FontBold	Marcar
		Text	Salvar
		TextColor	Red
		Rename	Btn_Salvar

A figura a seguir exibe o layout que utilizaremos durante este capítulo.

Figura 9.7: Layout da inclusão dos dados

Precisamos ainda inserir os componentes não visíveis do

projeto. Como vimos em capítulo anterior, para exibir as mensagens que o usuário receberá após cada solicitação de ação, vamos inserir um componente `Notifier` .

É necessário também um componente que faça a ligação do nosso aplicativo com a internet, enviando e recebendo as informações processadas. Vá até a guia de componentes `Connectivity` e insira o componente `Web` , responsável por esse processo. Renomeie-o para `Web_Cadastrar` , pois sua função será exclusivamente controlar os dados durante o processo de cadastro de informações no banco de dados. A figura a seguir exibe os componentes não visíveis que acabamos de inserir no projeto.

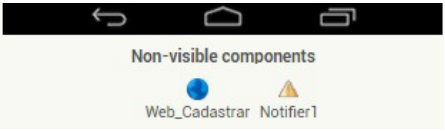

Figura 9.8: Componentes não visíveis

9.2 ARQUIVO GRAVAR.PHP

Antes de iniciar a parte da programação dos blocos para o cadastro dos dados, verificaremos a lógica usada para o aplicativo se conectar ao banco de dados em `MySQL` .

Depois de digitar todas as informações da corrida nas `TextBoxs` e clicar no botão `Salvar` , ativaremos o componente `Web_Cadastrar` que enviará as informações digitadas no aplicativo para um arquivo externo escrito na linguagem PHP. Este, por sua vez, receberá as informações e preparará os dados para o envio e gravação no banco de dados, armazenando-as.

Após a tentativa de gravação, o arquivo em PHP devolverá

uma informação ao aplicativo para que este possa verificar se a solicitação de cadastro foi efetuada com sucesso ou não. A seguir, vemos os códigos do arquivo `gravar.php` que realizam as tarefas de gravar as informações no banco de dados.

```
1.  <?php
2.  $nome =$_POST['nome'];
3.  $local =$_POST['local'];
4.  $distancia =$_POST['distancia'];
5.  $data= date('Y-d-m', strtotime($_POST['data']));
6.  $largada =$_POST['largada'];
7.  $conexao = mysql_connect('localhost','root','usbw');
8.  mysql_select_db('CORRIDA',$conexao);
9.  $sql = "insert into AGENDA (Nome, Local, Distancia, Data, Lar
gada) values ('$nome','$local','$distancia','$data','$largada')";
10. $resposta = mysql_query($sql) or die ("Erro: " . mysql_error(
));
11. if($resposta)
12.          echo "1";
13. else
14.       echo "0";
15. ?>
```

> Os arquivos em PHP também estão disponíveis para download no site do livro, mas caso o leitor queira digitar os códigos, utilize um editor de textos simples, como o Notepad++. Não digite o número das linhas, pois eles servem apenas para a explicação dos códigos.

Como o objetivo central deste livro é o App Inventor e não a linguagem PHP, vamos comentar brevemente o que cada linha de comando está realizando, pois precisamos da interação entre as duas tecnologias.

Analisando o código do arquivo `gravar.php`, temos:

- Linha 1 — Tag que indica a abertura de um arquivo em PHP.
- Linhas 2, 3, 4, 5 e 6 — Através do método `$_POST`, o PHP recebe as informações vindas do App Inventor e armazena-as nas variáveis locais dentro do PHP. São elas: `$nome`, `$local`, `$distancia`, `$data` e `$largada`.
- Linha 7 — Estamos definindo uma variável chamada `$conexao`. Sua função é realizar a conexão com o phpMyAdmin para depois acessar o banco de dados. Veja que, após o comando `mysql_connect`, são informados os valores: localhost, o nome do usuário e a senha de acesso ao phpMyAdmin. Se o leitor estiver utilizando outro servidor e não o *USBWebserver*, fique atento ao nome do usuário e à senha que deverão ser usados para cada servidor.
- Linha 8 — Realiza a identificação do banco de dados que utilizaremos no projeto, ou seja, o `CORRIDA`.
- Linha 9 — Estamos atribuindo para a variável `$sql` um comando para inserir os dados na tabela `AGENDA`.
- Linha 10 — É nessa linha que acontece a gravação dos dados no banco pelo comando `mysql_query($sql)`. Caso ocorra algum erro, será informado.
- Linhas 11 a 14 — Verifica se o resultado dos comandos da linha 10 foi concluído com êxito por meio do comando de decisão `if`. Caso as informações tenham sido gravadas no banco, será exibido como resposta o número **1**; caso ocorra

algum problema com a gravação dos dados, será exibido o número **0**. É essa informação que retornará para o aplicativo que estamos desenvolvendo e, através desse número, saberemos se as informações foram salvas ou não no banco de dados.

- Linha 15 — Encerra o bloco da estrutura de comandos do PHP.

Precisamos salvar o arquivo `gravar.php` dentro de uma pasta no diretório principal do servidor web local. Abra o diretório `root`, que é o principal, clicando no botão `Root dir` do aplicativo USBWebserver. A próxima imagem mostra a tela principal do USBWebserver com esse botão.

Figura 9.9: Botão Root dir

Será exibida a janela do Windows Explorer. Nela crie uma nova pasta e dê o nome de `app_run`. Veja como ficará a estrutura de suas pastas depois disso:

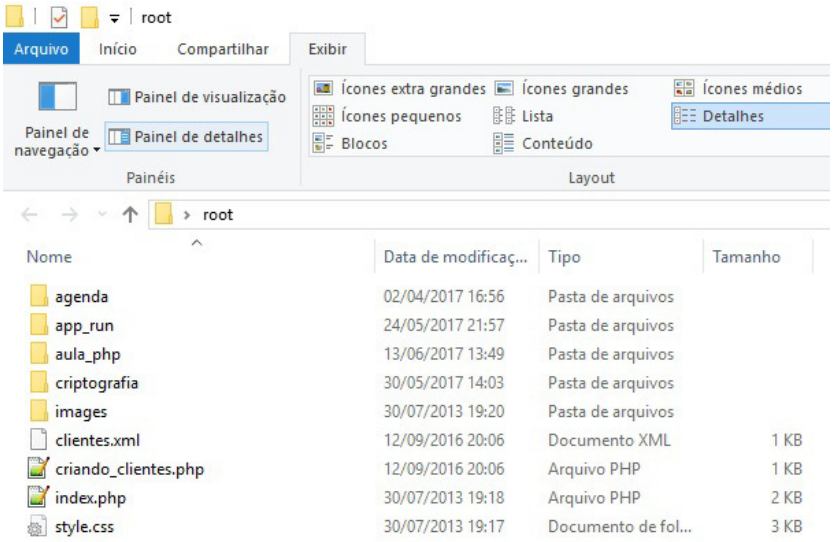

Figura 9.10: Pasta app_run

É dentro da pasta `app_run` que salvaremos o arquivo `gravar.php`. Veja onde ficará seu arquivo depois de salvá-lo:

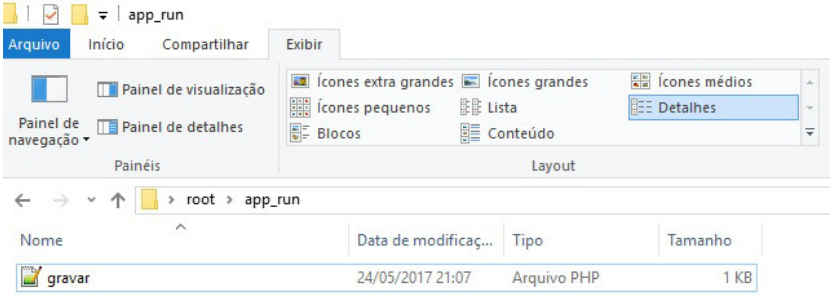

Figura 9.11: Arquivo gravar.php salvo

9.3 BLOCOS PARA A INCLUSÃO

Agora podemos voltar à tela do App Inventor e acessar a guia

`Blocks` para programarmos os componentes.

É possível criar cores diferentes das disponíveis na tela de design. Esse processo de criação consiste basicamente em criar uma variável e colocar um bloco para definir os valores numéricos da cor.

Acesse a área `Built-in` e, na seção `Variables`, insira um bloco da opção `initialize global name to` em sua área `Viewer`. Agora, na seção `color`, localize a opção `make color` e encaixe-a na criação da variável. Então, é só definir o número da composição das cores no padrão RGB (*Red, Green e Blue*).

Nomeie essa variável para `fundo`, porque a cor que estaremos definindo será atribuída ao fundo do aplicativo. Nos valores das cores, informe os números 151, 161, 127. Seguindo esse procedimento, crie uma outra variável de cor chamada `titulo` e defina os seguintes valores para ela: 17, 85, 17. A próxima figura mostra as variáveis `fundo` e `titulo`.

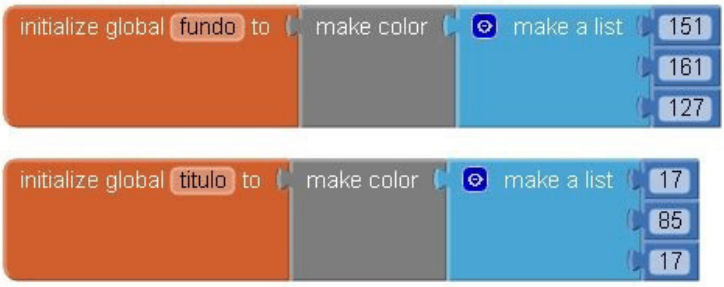

Figura 9.12: Criando novas cores

Após a criação das cores, precisamos configurar onde e quando serão exibidas. Quando inicializarmos o aplicativo, elas terão de ser

ativadas imediatamente. Existe um bloco que realiza tarefas quando o aplicativo for iniciado, trata-se do `when Screen1.initialize`.

Vamos primeiramente acessar a área `Blocks` e selecionar o componente `Screen1` para expandir seus blocos de controle. Localize e insira o bloco `when Screen1.Initialize`. Dentro dele, devemos incluir os componentes que receberão as novas cores criadas. Selecione e insira um bloco de `set Screen1.BackgroundColor to` e um bloco `set Horizontal_Titulo.BackgroundColor to`.

Na sequência, encaixe as variáveis referentes às cores criadas. Esse procedimento alterará as cores de fundo da `Screen1` e do `Horizontal_Titulo` quando o app for inicializado. Verifique o resultado dos blocos:

when Screen1 .Initialize
do set Screen1 . BackgroundColor to get global fundo
set Horizontal_Titulo . BackgroundColor to get global titulo

Figura 9.13: Utilizando as cores

Precisamos configurar a exibição da data no aplicativo, pois o objeto `DatePicker` apenas habilita a apresentação do calendário. Insira o bloco do componente `When DatePicker1.AfterDateSet`. Ele indica que, após a seleção da data, precisamos configurá-la para exibição no componente `Txt_Data.Text` que está na sua `Screen1` — este indica a **data** em que ocorrerá a corrida.

Insira um bloco `set Txt_Data.Text to` no interior do bloco

`When DatePicker1.AfterDateSet` . Para que a data seja exibida no formato **dd/mm/aaaa**, precisamos *juntar* as informações do dia, mês e ano que são identificadas isoladamente pelo componente `DatePicker1` . Para concatená-las, precisamos de um bloco de texto `join` . Nele, acrescente mais três blocos de texto para definirmos o dia, o mês e o ano separados por um bloco de texto com barra (`/`).

Localize e selecione na guia `Blocks` o componente `DatePicker1` . Para inserir o dia selecionado pelo usuário, é necessário escolher a opção `DatePicker1.Day` e encaixar na primeira opção da `join` . Na segunda opção, insira um bloco de texto e digite uma barra `/` em seu interior. Para inserir o mês e o ano selecionados, usaremos o mesmo procedimento para a seleção do dia, porém com os blocos `DatePicker1.Month` e `DatePicker1.Year` .

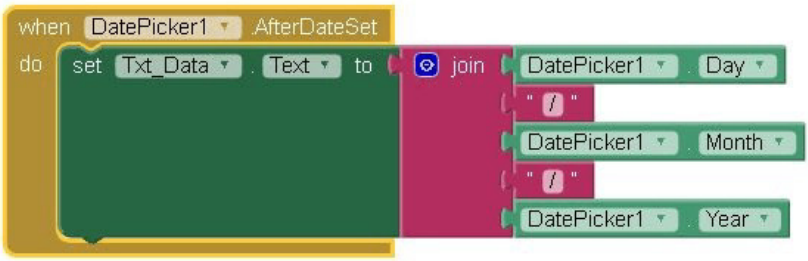

Figura 9.14: Configurando a exibição da data

Da mesma maneira como fizemos a configuração do `DataPicker1` , devemos realizar a configuração do horário de largada da corrida. Acrescente um bloco do `When TimePicker1.AfterTimeSet` .

Antes de realizarmos as configurações do horário, precisaremos fazer um ajuste para a exibição de dois dígitos quando os minutos forem na faixa de 0 a 9. Para tanto, é preciso criar uma variável com o nome `AjustarMinutos` e encaixar um bloco de texto vazio.

Agora, dentro do bloco `When TimePicker1.AfterTimeSet`, necessitamos verificar se os minutos estão nessa faixa de valor utilizando o bloco de controle `IF` configurado com uma opção `else`. Caso os minutos estejam, devemos configurar a variável `AjustarMinutos` para receber pelo bloco de comando `Join` um bloco de texto com o valor `0`, juntamente com o valor dos minutos indicados pelo bloco `TimerPicker1.Minute`. Para a opção do `else`, ajuste a variável `AjustarMinutos` com o próprio valor do `TimerPicker1.Minute`.

Para a exibição dos valores do horário na tela do aplicativo, devemos ajustar o bloco de texto `Set Txt_Horario.Text to` com um bloco de texto `join`, configurando-o para receber mais uma opção de texto. Veja na próxima figura o resultado do bloco da configuração no formato `hh:mm`.

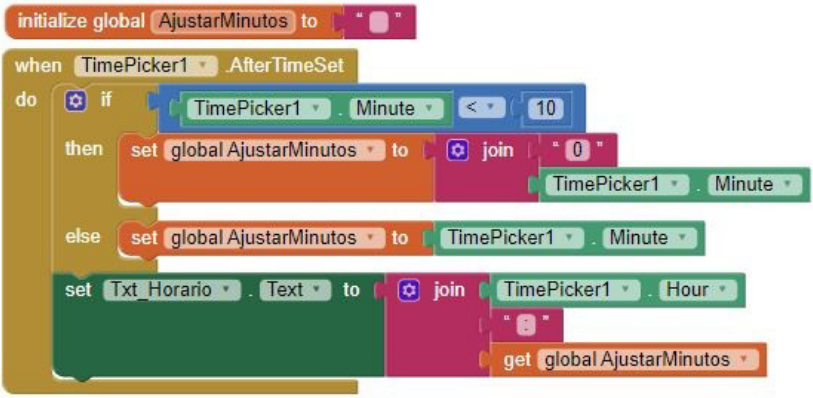
initialize global AjustarMinutos to " "
when TimePicker1 .AfterTimeSet
do if TimePicker1 . Minute < 10
then set global AjustarMinutos to join " 0 "
TimePicker1 . Minute
else set global AjustarMinutos to TimePicker1 . Minute
set Txt_Horario . Text to join TimePicker1 . Hour
" : "
get global AjustarMinutos

Figura 9.15: Configurando a exibição da hora

9.4 A LÓGICA DO BOTÃO SALVAR

Para um bom aproveitamento de estudos, vamos primeiro conhecer os passos lógicos que estão por trás do clique do botão que realizará a gravação das informações no banco de dados:

1. Temos de verificar se todas as informações foram digitadas para podermos prosseguir. Avisaremos ao usuário se ele se esqueceu de digitar algo.

2. Indicaremos para o componente `Web_Cadastrar` o caminho de onde o arquivo `gravar.php` está salvo.

3. Diremos quais informações serão encaminhadas para o arquivo externo `gravar.php` , para que ele possa processar e salvar no banco de dados.

4. Após gravar as informações, devemos limpar as informações que foram digitadas na tela.

Vamos programar o bloco `When Btn_Salvar.Click`, pois esse é o botão responsável por receber os dados que você digitou e inseri-los no banco. Insira um bloco `When Btn_Salvar.Click` na área de `Viewer` e, logo após, insira um bloco de decisão `If`, configurando-o para possuir a opção `else`.

Figura 9.16: Configurando a exibição da hora

Para verificar se todas as informações foram digitadas, necessitamos realizar um teste para cada uma das caixas de texto. Temos de trabalhar com o comando lógico `OR`. Como temos 5 caixas de texto para verificar, insira quatro blocos `OR` da seção `Logic` e encaixe um dentro do outro, conforme demonstrado na próxima figura.

Figura 9.17: Blocos lógicos encaixados

Precisamos verificar se cada uma das caixas de texto está vazia, isto é, sem valor digitado. É necessário preparar os blocos para essa verificação. Selecione a seção `Text` na `Built-in` e insira um controle de texto que verifica se o campo está vazio (`is empty`) e,

na sequência, encaixe o controle do componente `Txt_Corrida.Text` . Repita esse procedimento para os outros controles de entrada. A figura a seguir exibe os blocos de verificação ainda não posicionados.

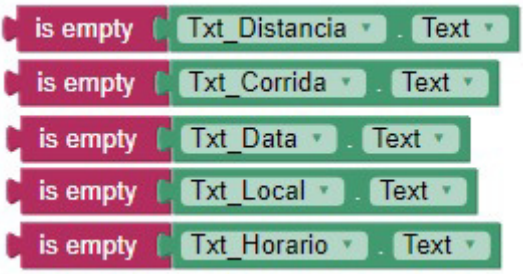
is empty Txt_Distancia . Text
is empty Txt_Corrida . Text
is empty Txt_Data . Text
is empty Txt_Local . Text
is empty Txt_Horario . Text

Figura 9.18: Blocos is empty

Necessitamos encaixar cada bloco criado dentro dos espaços deixados no bloco `IF` . Caso uma das comparações for verdadeira — isto é, caso o usuário não tenha digitado alguma das informações —, o aplicativo emitirá a mensagem `Digite todos os valores` pelo componente `notifier` . Para exibir a mensagem de alerta, insira um `Call Notifier1.ShowAlert` logo após a opção `Then` do bloco `IF` , e encaixe um bloco de texto com a informação: `Digite todos os valores` . Até o momento, temos:

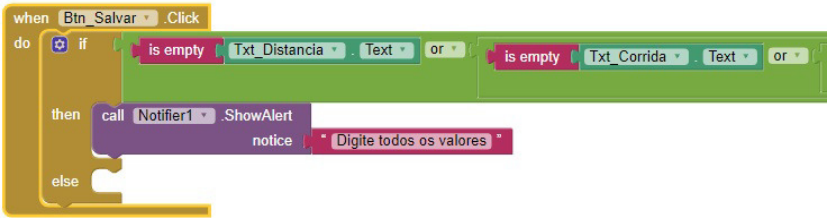
when Btn_Salvar .Click
do if is empty Txt_Distancia . Text or is empty Txt_Corrida . Text or
then call Notifier1 .ShowAlert
notice " Digite todos os valores "
else

Figura 9.19: Btn_Salvar

Caso todas as informações tenham sido digitadas, precisamos informar ao componente `Web_Cadastrar` qual o endereço do local onde está salvo o arquivo `gravar.php`. Para que isso ocorra, insira o bloco `set Web_Cadastrar.Url to` na frente da opção `else` do bloco `IF`, e complete o endereço com uma caixa de texto.

Nesse bloco de texto, insira o seu número IP (aquele que identificamos no capítulo anterior) no seguinte formato: **http://seu.numero.ip:8080/app_run/gravar.php**, em que `8080` é o número da porta na qual o servidor web local (USBWebserver) está habilitado. Se você alterou esse valor, insira o valor correto. O `app_run` é a pasta onde salvamos o arquivo `gravar.php`, conforme visto neste capítulo.

A figura a seguir exibe o bloco `Web_Cadastrar.Url` configurado com o endereço do arquivo `gravar.php`:

Figura 9.20: Bloco Web_Cadastrar.Url configurado

Após indicar o local do arquivo `gravar.php`, teremos de informar os dados que serão enviados para o arquivo em PHP. Insira o bloco do componente que realiza este envio: `call Web_Cadastrar.PostText`.

Como teremos várias informações para envio, necessitamos de um bloco de texto `join`. Configure a `join` para receber, além das duas entradas de texto já exibidas como padrão, mais 8 opções de encaixe. Para cada opção, precisaremos ter um bloco de texto com uma informação digitada e, na linha abaixo, um bloco do

componente que contenha essa informação.

A tabela a seguir exibe a ordem e os valores que devem ser encaixados nas opções:

Informação	Componente
nome=	Bloco de texto
Txt_Corrida.Text	Bloco de componente
&local=	Bloco de texto
Txt_Local.Text	Bloco de componente
&distancia=	Bloco de texto
Txt_Distancia.Text	Bloco de componente
&data=	Bloco de texto
Txt_Data.Text	Bloco de componente
&largada=	Bloco de texto
Txt_Horario.Text	Bloco de componente

O resultado do bloco configurado anteriormente será uma informação com as seguintes características:

```
nome=Txt_Corrida.Text&local=Txt_Local.Text&distancia=Txt_D
istancia.Text&data=Txt_Data.Text&largada=Txt_Horario.Text
```

Todos esses dados serão enviados para o arquivo `gravar.php` poder incluí-los no banco de dados.

A figura a seguir exibe o bloco `call Web_Cadastrar.PostText` configurado para o envio das

informações.

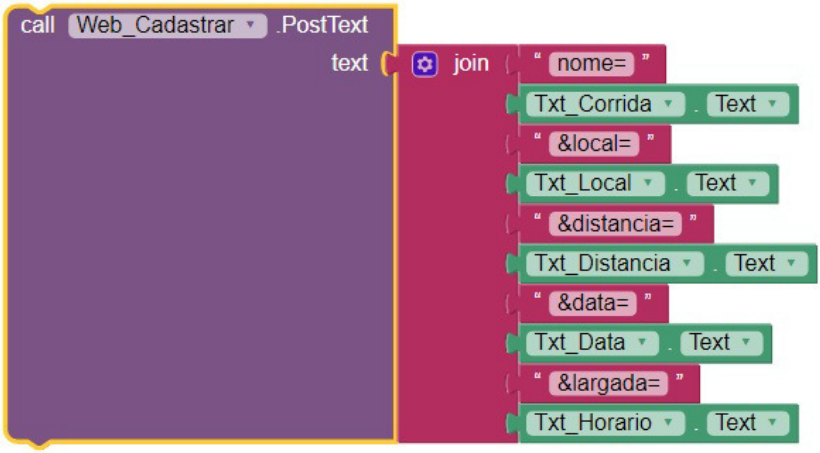

Figura 9.21: Bloco call Web_Cadastrar.PostText finalizado

Após o envio das informações, devemos apagar todos os valores que já foram usados e ainda estão visíveis nas `TextBoxs`. Vamos inserir um primeiro bloco `set Txt_Corrida.Text to` e encaixar um bloco de texto vazio para remover o valor digitado. Repita essa operação para as demais `TextBoxs` de sua tela. Veja como o `Btn_Salvar` ficará quando finalizado:

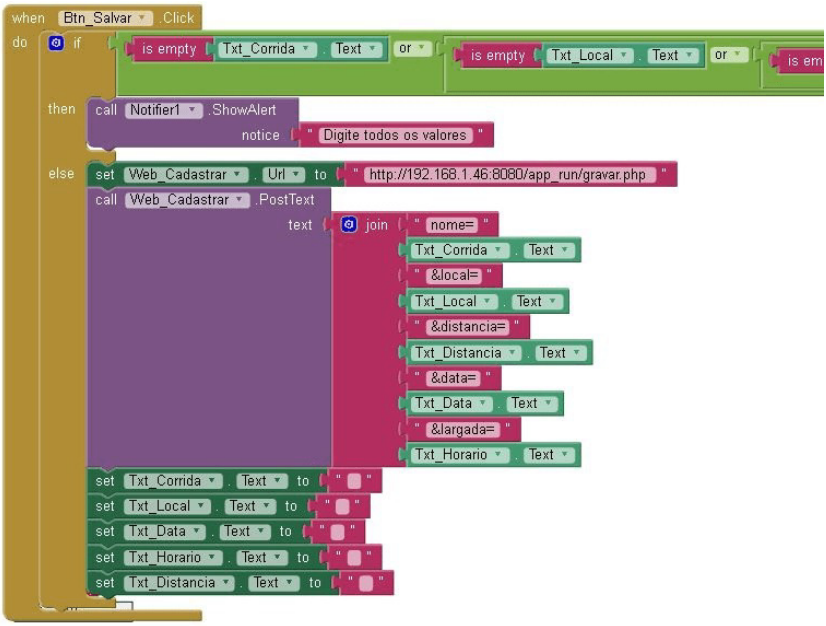
when Btn_Salvar .Click
do if is empty Txt_Corrida . Text or is empty Txt_Local . Text or is em
then call Notifier1 .ShowAlert
notice " Digite todos os valores "
else set Web_Cadastrar . Url to " http://192.168.1.46:8080/app_run/gravar.php "
call Web_Cadastrar .PostText
text join " nome= "
Txt_Corrida . Text
" &local= "
Txt_Local . Text
" &distancia= "
Txt_Distancia . Text
" &data= "
Txt_Data . Text
" &largada= "
Txt_Horario . Text
set Txt_Corrida . Text to " "
set Txt_Local . Text to " "
set Txt_Data . Text to " "
set Txt_Horario . Text to " "
set Txt_Distancia . Text to " "

Figura 9.22: Bloco Btn_Salvar finalizado

9.5 RECEBENDO E VERIFICANDO OS DADOS DE RESPOSTA

Como vimos, o comando `call Web_Cadastrar.PostText` enviou as informações digitadas para o arquivo `gravar.php` , que as recebeu e tentou inserir no banco de dados. Como resposta a essa solicitação, temos duas possibilidades: sucesso ou não. Precisamos verificar o que ocorreu.

Conforme as linhas 11, 12, 13 e 14 do arquivo em PHP visto anteriormente, será retornado para o aplicativo o valor `1` se tudo der certo, ou o número `0` em caso de erro. Precisaremos verificar qual receberemos. O componente `Web_Cadastrar` possui um

evento que recebe um texto de retorno do arquivo em PHP, trata-se do `GotText`.

Na seção dos blocos, selecione o componente `Web_Cadastrar` e localize o bloco `When Web_Cadastrar.GotText`. Insira-o na área de `Viewer`.

Dentro dele, para realizarmos a verificação do valor retornado, devemos inserir um bloco de controle `if` com a opção `else` configurada. Veja na figura a seguir como deverá estar o bloco `When Web_Cadastrar.GotText` nesse momento.

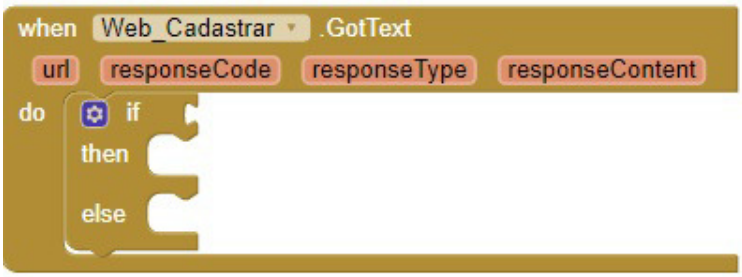

Figura 9.23: Bloco When Web_Cadastrar.GotText

Como precisamos verificar se a informação retornada **contém um texto**, é necessário inserir o bloco que representa essa informação. Insira, da guia `Text`, um bloco `contains text` e encaixe-o no bloco `if`.

Conseguimos receber a informação obtida como resposta do arquivo `gravar.php` através do bloco de comando `get responseContent`, que você encontra no bloco `Web_Cadastrar.GotText` apenas descansando o ponteiro do mouse sobre a opção `responseContent` e arrastando-o para encaixar logo após o `contains text`. Como queremos verificar

se a resposta enviada pelo arquivo é o número `1`, devemos encaixar um bloco de texto na opção `piece` e, dentro dele, digitar `1`.

Caso a resposta seja esse número, devemos informar ao usuário que a informação foi inserida com sucesso no banco de dados. Para isso, depois da opção `then` do bloco de controle `if`, insira um bloco do componente `call notifier1.ShowAlert`, encaixe um bloco de texto e, dentro dele, digite **Gravado com sucesso!**.

Caso tenha ocorrido algum erro durante a tentativa de salvar a informação, a resposta obtida pelo aplicativo será o número `0`. Então, devemos programar os blocos após a opção `else` do bloco `if`, inserindo um outro bloco do componente `call notifier1.ShowAlert` e encaixando um bloco de texto. Nele digite a informação **Erro ao salvar!**.

Veja como ficará o seu bloco `Web_Cadastrar.GotText`:

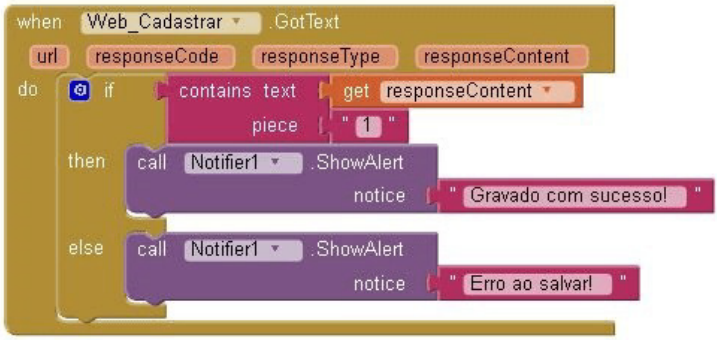

Figura 9.24: Bloco When Web_Cadastrar.GotText

9.6 EMULANDO PARA SALVAR AS

INFORMAÇÕES

Vamos emular nosso aplicativo diretamente em seu computador e ver como deverá ser seu comportamento. Aguarde o carregamento total de seu aplicativo:

Figura 9.25: Aplicativo sendo emulado

Preencha as informações nas caixas de texto com os dados referentes ao nome da corrida, localização, distância do percurso, data do evento e horário da largada. Depois, clique no botão `Salvar` . Confira na figura a seguir um exemplo com os valores digitados.

Figura 9.26: Dados digitados no aplicativo

Espere pela mensagem de **Gravado com sucesso!**. Caso a mensagem seja **Erro ao salvar!** ou outra qualquer, é preciso verificar todos os passos vistos anteriormente, como, por exemplo, se o servidor *USBWebserver* está ativado ou se os nomes do banco de dados, da tabela e dos campos estão escritos corretamente. Vale lembrar que há diferenças entre escrever um campo com letras maiúsculas e utilizar letras minúsculas, ou vice-versa.

Figura 9.27: Bloco When Web_Cadastrar.GotText

Após a mensagem de **Gravado com sucesso!** ser exibida, volte até a tela do phpMyAdmin e veja se os dados digitados estão salvos em nossa tabela. A próxima figura exibe a tela do phpMyAdmin com a informação que foi salva pelo aplicativo. Como sugestão, insira mais corridas para utilização no próximo capítulo.

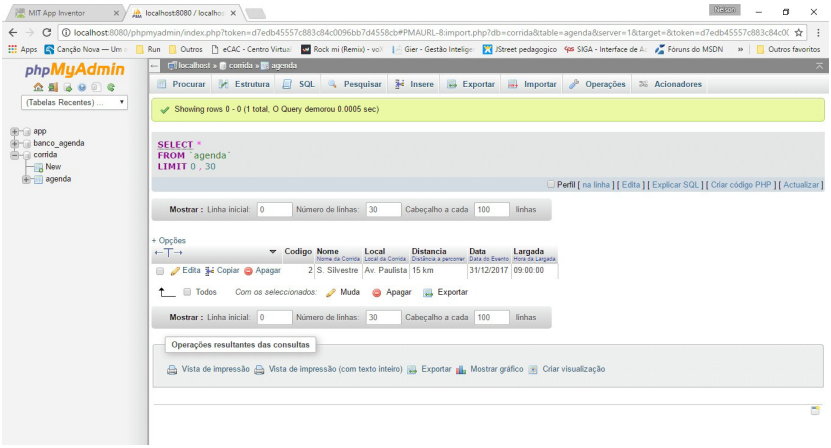

Figura 9.28: Tela do phpMyAdmin com a informação salva

9.7 RESUMINDO

Neste capítulo, criamos o layout do aplicativo de corridas, vimos o arquivo em PHP e configuramos o botão para enviar as informações digitadas para o banco de dados por meio desse arquivo. Essas informações estarão disponíveis para consulta nas próximas vezes que você executar seu aplicativo.

No próximo capítulo, veremos o botão de consulta, como localizar as informações do banco de dados e exibi-las na tela de seu aplicativo.

CAPÍTULO 10

CONSULTANDO O BANCO DE DADOS

Agora que já digitamos as informações da corrida no aplicativo e salvamos no banco de dados em MySQL, vamos realizar uma consulta no banco, listando todos os nomes das corridas cadastradas e, após escolher uma delas, exibir as demais informações na tela do app.

Conheceremos o componente `ListPicker` que se encontra na guia `User Interface` . Ele será responsável por armazenar os dados vindos do banco e pela exibição em tela nos campos específicos.

10.1 ADICIONANDO OS COMPONENTES DA CONSULTA

Vamos retornar à área de `Design` do projeto de `minhas_corridas` para inserir mais três componentes na tela do aplicativo. Veja na tabela a seguir os novos componentes que deverão ser inseridos e suas propriedades que serão alteradas.

Componente	Guia	Propriedade	Alterar valor para
ListPicker	User Interface	Visible	Desmarcar

		Rename	List_Corrida
Button	User Interface	FontBold	Marcar
		FontSize	16
		Text	Listar Corridas
		TextColor	Red
		Rename	Btn_Listar
Web	Connectivity	Rename	Web_Listar

Inserimos um componente `ListPicker`, que será responsável por exibir uma lista com os nomes de todas as corridas cadastradas anteriormente. Ele ficará invisível na tela de design, pois sua visualização será ativada após o usuário clicar no botão `Btn_Listar`, como veremos adiante. Alteramos o nome do componente para `List_Corrida` para uma melhor identificação.

Configuramos o `Button` para exibir o texto `Listar Corrida`, alteramos o tamanho da fonte para `16`, selecionamos o efeito negrito e deixamos a cor do texto como vermelho para melhor identificação do botão e de sua função, que será **Listar todos os nomes das corridas cadastradas**.

O componente `Web` ficará na área `Non-visible components` e terá a mesma função apresentada no capítulo anterior: enviar os dados de solicitação para o arquivo `listar.php`. Alteramos seu nome da `Web_Listar`.

Veja na figura a seguir o layout atualizado, após inserção dos componentes:

Figura 10.1: Atualização do layout

10.2 ARQUIVO LISTAR.PHP

Antes de programar os blocos, vamos entender como será o processo para recuperar os nomes de todas as corridas cadastradas no banco de dados. Quando o usuário clicar no botão `Btn_Listar`, este vai configurar o endereço do local onde se encontra o arquivo `listar.php` pelo componente `Web_Listar`. É esse arquivo que realizará uma consulta no banco de dados e retornará as informações para o nosso app.

Veja a seguir os comandos do arquivo `listar.php`. Abra seu editor de texto e digite os códigos:

```
1.  <?php
2.  $conexao = mysql_connect('localhost','root','usbw');
3.  mysql_select_db('CORRIDA',$conexao);
4.  $sql="select * from AGENDA order by Data desc";
5.  $resultado = mysql_query($sql) or die ("Erro: " . mysql_error
());
6.  while($linha = mysql_fetch_object($resultado))
7.      echo $linha->Nome."#";
8.  echo "^";
9.  ?>
```

Vamos entender o arquivo `listar.php` que realiza a consulta

apenas dos nomes das corridas cadastradas:

- Linha 1 — Tag que indica a abertura de um arquivo em PHP.
- Linha 2 — Estamos criando a variável `$conexao` com o phpMyAdmin, onde são informados o localhost, o usuário e a senha de acesso ao phpMyAdmin, como vimos no capítulo anterior.
- Linha 3 — Realiza a identificação do nome do banco de dados que vamos utilizar no projeto, ou seja, o `CORRIDA`.
- Linha 4 — Estamos definindo a instrução da linguagem SQL para a variável `$sql`, para consultar todas as informações do banco de dados.
- Linha 5 — É nessa linha que acontece a consulta dos dados no banco através do comando `mysql_query($sql)`. Caso ocorra algum erro, será informado ao usuário.
- Linhas 6 e 7 — Através do comando `While` (enquanto), o programa realiza a leitura dos registros que foram lidos da tabela `AGENDA`. Essa linha retornará para o aplicativo apenas o conteúdo do campo `Nome` da corrida, seguido de um símbolo `#` após cada nome de corrida. Esse símbolo (cerquilha) será utilizado para separar cada nome de corrida cadastrada e inseri-lo como lista de itens no `ListPicker` (`List_Corrida`), que veremos adiante.
- Linha 8 — Quando acabarem os nomes cadastrados na tabela, vamos inserir um acento circunflexo (`^`) para indicar ao App Inventor que os registros terminaram.

- Linha 9 — Encerra o bloco da estrutura de comandos do PHP.

Você deverá salvar esse arquivo com o nome `listar.php`. Assim como o arquivo `gravar.php`, usado no capítulo anterior, ele ficará gravado na mesma pasta, ou seja, dentro do diretório `root` do servidor *USBWebserver*, e dentro da pasta chamada `app_run`.

10.3 CRIANDO A LISTA DE CORRIDAS

Vamos voltar à tela de desenvolvimento do App Inventor e programar os blocos para realizar uma consulta geral de todas as corridas cadastradas quando clicarmos no componente `Btn_Listar`. Clique no botão `Blocks` para acionarmos a área de blocos.

Começaremos preparando a programação do botão `Btn_Listar`. Insira um bloco `when Btn_Listar.click`, ou seja, quando você clicar no componente `Btn_Listar`, ele executará as instruções definidas em seu interior.

Precisamos indicar para o componente `Web_Listar` o local onde se encontra o arquivo `listar.php`. Insira um `set Web_Listar.Url to` para encaixar nele um bloco de texto que receberá o seu número IP com o nome da pasta, seguido do nome do arquivo em PHP. No meu caso, digitei http://192.168.1.35:8080/app_run/listar.php, mas você, leitor, terá de digitar seu próprio número de IP.

Precisamos de um bloco que realize a chamada ao arquivo `listar.php` para obter os dados do processamento do PHP. Para

isso, coloque o bloco do componente `call Web_Listar.Get` . Na sequência, acrescente um bloco para esconder a visualização do teclado do dispositivo que ainda pode estar visível, o `call Txt_Local.HideKeyboard` . Ele deverá ser inserido logo abaixo do `call Web_Listar.Get` .

A figura a seguir exibe o bloco de programação do `Btn_Listar` finalizado:

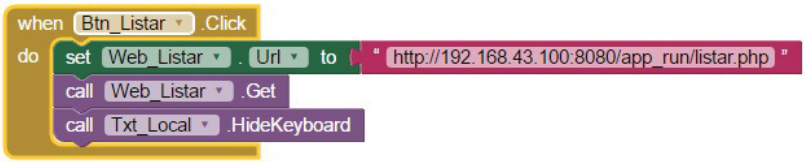
when Btn_Listar .Click
do set Web_Listar . Url to " http://192.168.43.100:8080/app_run/listar.php "
call Web_Listar .Get
call Txt_Local .HideKeyboard

Figura 10.2: Btn_Listar

Precisamos acrescentar uma variável chamada `separa` que terá como função separar os nomes das corridas recebidas do arquivo `listar.php` para posterior exibição na lista `List_Corrida` . Finalize a variável encaixando um bloco de texto vazio, pois ela armazenará um texto.

Figura 10.3: Variável separa

Depois do clique no botão `Btn_listar` , onde chamamos o método `get` (obter dados) do componente `Web_Listar` , ele receberá as informações enviadas pelo processamento do arquivo `listar.php` . Então, devemos separar cada nome de corrida que está marcado pelo símbolo `#` e exibi-los como uma linha da lista `List_Corrida` . Insira primeiramente o bloco do `when`

`Web_Listar.GotText` , pois é ele que trabalha com o texto obtido.

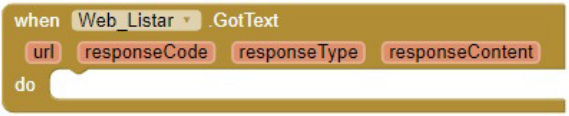

Figura 10.4: Web_listar.GotText

Em seu interior, para que possamos verificar se a consulta retornou algum registro do banco de dados, insira um bloco de controle `if` e configure-o para que possua a opção `else` . Assim, vamos preparar a condição de decisão que teremos no comando `if` .

Como é preciso verificar se o texto recebido do arquivo `listar.php` é igual ao caractere que definimos para identificar que não teríamos nenhuma corrida cadastrada (o símbolo `^`), devemos selecionar um bloco de comparação com o sinal de `=` (igual) da guia `Built-in` na seção `Logic` , e encaixá-lo no bloco `if` .

No primeiro espaço livre, encaixamos o bloco que recebe as informações vindas do arquivo `listar.php` , o `get responseContent` . Para localizá-lo, descanse o ponteiro do mouse sobre a opção `responseContent` dentro do `when Web_listar.GotText` , selecione e arraste-o para dentro da comparação.

Para verificar se a informação recebida foi o `^` , necessitamos inserir um bloco de texto e digitar esse símbolo. Essa comparação verifica se foi retornado **apenas** esse símbolo e, se isso ocorrer, quer dizer que não temos nenhuma corrida cadastrada e devemos

exibir essa informação ao usuário. Para isso, após o `then`, insira um bloco `call Notifier1.ShowAlert` e nele encaixe um bloco de texto na opção `notice` com a informação: **Nenhuma corrida cadastrada :-(**. Veja o resultado:

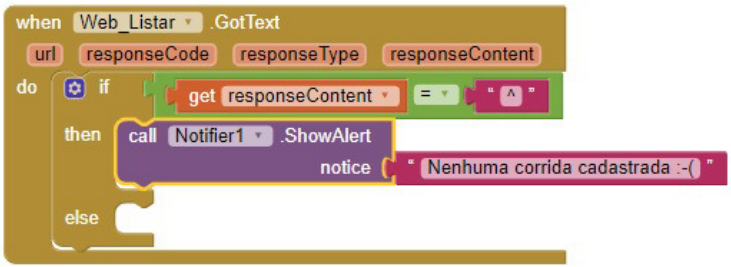

Figura 10.5: Nenhuma corrida cadastrada

Agora vamos trabalhar com a opção `else`, pois se não recebermos apenas o caractere `^`, é porque existem corridas cadastradas na tabela e elas deverão ser exibidas no componente `List_Corrida`.

Você se lembra do caractere `#` que inserimos no arquivo `listar.php`? É ele que nos auxiliará na tarefa de separar os nomes das corridas, pois o retorno das informações do arquivo PHP é recebido conforme o exemplo: `S. Silvestre#Maratona de SP#Meia de Florianópolis#^`.

Agora vamos separar os nomes das corridas. Vamos procurar por todo o texto que recebemos e trocar o `#` pela `,` (vírgula), pois é apenas com ela que o componente `List_Corrida` saberá que deve exibir a informação em uma nova linha, criando uma lista com todas as corridas.

Vamos realizar essa troca de caracteres e armazenar todo o

resultado obtido na variável `separa`, que criamos anteriormente. Insira o bloco da variável `set global separa to` e encaixe-o logo após a opção `else` do `if`.

Para realizar a substituição de símbolos, temos de inserir um bloco da guia `Built-in` da seção `text`. Localize o bloco `replace all text ... segment .... replacement` e insira depois da variável `separa`.

Como precisamos localizar todo o texto que contenha o símbolo `#`, necessitamos inserir mais um bloco `get responseContent` e encaixá-lo na opção que localiza no texto todo, ou seja, no `replace all text`. Devemos agora indicar **qual** texto deverá ser localizado para ser substituído. Então, em `segment`, insira um bloco de texto e digite o símbolo `#` no seu interior.

Finalmente, precisamos indicar que o caractere `#` deverá ser substituído pela vírgula. Em `replacement`, insira um bloco de texto com o símbolo `,` em seu interior. Com essa operação, o resultado do exemplo citado ficaria: `S. Silvestre,Maratona de SP,Meia de Florianópolis,^`.

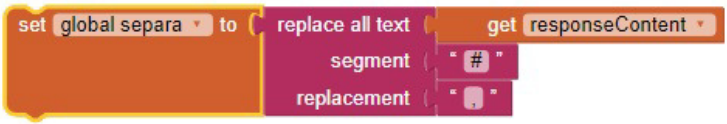

Figura 10.6: Substituição de caracteres

Conforme visto no resultado da troca dos caracteres do exemplo, ainda temos o sinal que indica o final dos registros, o `^`. Precisamos retirá-lo para que não apareça na lista de corridas do

aplicativo. O procedimento adotado para isso é praticamente o mesmo visto na troca do `#` pela `,`.

Então, insira mais um bloco da variável `set global separa to` e, na sequência, arraste mais um bloco de texto `replace all text ... segment .... replacement`. O texto já modificado está armazenado na variável `separa`, e é o bloco da variável `get global separa` que deverá ser encaixado na opção que localiza o texto para ser substituído (`replace all text`).

Em `segment`, coloque um bloco de texto com o sinal a ser localizado para a substituição. Para a opção `replacement`, insira um bloco de texto vazio. Assim, retiramos o `^` final do arquivo e não vamos inserir nada em seu lugar.

Com essa operação, o exemplo citado ficaria: `S. Silvestre,Maratona de SP,Meia de Florianópolis,`. A figura a seguir exibe o bloco que realiza essa substituição.

Figura 10.7: Retirando o marcador de final do arquivo

A variável `separa` está armazenando os nomes das corridas seguidas por uma vírgula (`,`) e, para criar a lista, devemos posicioná-la no componente `List_Corrida`. Na área dos blocos, ache o componente `List_Corrida` e selecione a opção `set List_Corrida.ElementsFromString to`. Depois, selecione o bloco da variável `get global separa` para exibir os dados das corridas, encaixando-o no `List_Corrida`.

Finalmente, devemos exibir a lista de corridas na `ListPicker`. Na área dos blocos, localize o componente `List_Corrida` e selecione o bloco que exibe a lista com os dados da corrida já configurada, o `call List_Corrida.Open`. A figura exibe o componente `when Web_Listar.GotText`:

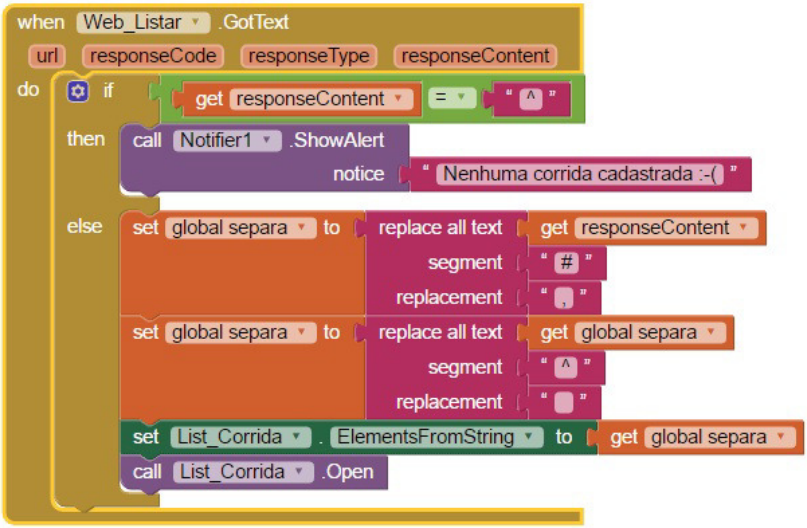
when Web_Listar .GotText
url responseCode responseType responseContent
do if get responseContent = " ^ "
then call Notifier1 .ShowAlert
notice " Nenhuma corrida cadastrada :-("
else set global separa to replace all text get responseContent
segment " # "
replacement " , "
set global separa to replace all text get global separa
segment " ^ "
replacement " "
set List_Corrida . ElementsFromString to get global separa
call List_Corrida .Open

Figura 10.8: When Web_Listar.GotText finalizado

10.4 EXIBINDO A LISTA DE CORRIDAS

Emule o seu projeto e, quando estiver carregado, clique no botão `Listar Corrida` para ver a lista de corridas que estão cadastradas. A figura a seguir exibe a tela com uma lista. As corridas são ordenadas pela data em que ocorrerão, devido à instrução da quarta linha do arquivo `listar.php`, visto anteriormente.

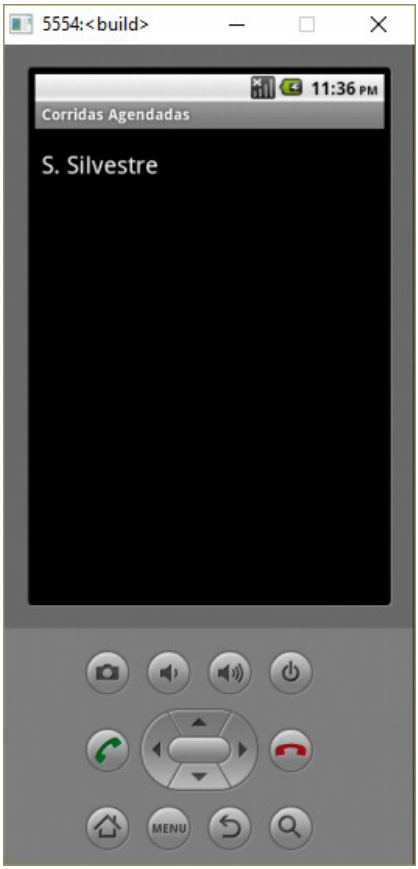

Figura 10.9: Exibição da lista

10.5 EXIBINDO TODOS OS CAMPOS DA CORRIDA

Após a exibição da lista com os nomes das corridas que estão no banco de dados, precisamos que sejam exibidas todas as informações complementares da corrida que será selecionada pelo usuário.

Devemos inserir em nossa tela de design mais dois componentes responsáveis por receber todos os campos do banco de dados e realizar a conexão com o arquivo `selecionar.php`, que veremos adiante. Insira os componentes conforme as configurações:

Componente	Guia	Propriedades	Alterar valor para
ListPicker	User Interface	Visible	Desmarcar
		Rename	List_Selecionar
Web	Connectivity	Rename	Web_Selecionar

O componente `ListPicker` teve seu nome alterado para `List_Selecionar` e foi marcado como invisível durante a execução do app. Sua finalidade será guardar os detalhes da corrida (nome, distância, local, data e horário) e, a partir do `List_Selecionar`, será exibida cada informação nas `TextBoxs` específicas da tela do aplicativo. Para o componente `Web`, alteramos o seu nome para `Web_Selecionar`.

Antes de programarmos os próximos blocos, precisamos produzir o arquivo `selecionar.php`, que vai realizar a tarefa de buscar as demais informações da corrida selecionada no banco de dados. A seguir, temos os comandos do arquivo `selecionar.php`. Abra seu editor de texto e digite os códigos exibidos, ou baixe-os em http://nelfabbri.com/appinventor/appinventor.zip.

```
1.  <?php
2.  $nome=$_POST['nome'];
3.  $conexao = mysql_connect('localhost','root','usbw');
4.  mysql_select_db('CORRIDA',$conexao);
5.  $sql="select *, DATE_FORMAT(Data, '%d/%m/%Y') as 'Data' from
AGENDA where Nome like '$nome'";
```

```
6.  $resultado = mysql_query($sql) or die ("Erro: " . mysql_error
());
7.  while($linha = mysql_fetch_object($resultado)) {
8.    echo $linha->Nome."#";
9.      echo $linha->Local."#";
10.      echo $linha->Distancia."#";
11.      echo $linha->Data."#";
12.      echo $linha->Largada."#";
13.      }
14.   echo "^";
15.   ?>
```

Vamos às explicações sobre o arquivo `selecionar.php`:

- Linha 1 — Tag que indica a abertura de um arquivo em PHP.
- Linha 2 — Através do método `$_POST`, estamos recebendo a informação vinda do App Inventor e armazenando-a na variável local `$nome`.
- Linha 3 — Estamos criando a variável `$conexao` que é a responsável por se conectar ao phpMyAdmin. Nela são informados o localhost, o usuário e a senha de acesso ao phpMyAdmin, como já vimos no capítulo anterior.
- Linha 4 — Realiza a identificação do banco de dados que vamos utilizar no projeto, ou seja, o banco `CORRIDA`.
- Linha 5 — Estamos atribuindo para a variável `$sql` um comando da linguagem **SQL** para consultar todos os campos do banco referentes ao **nome** da corrida escolhida pelo usuário do aplicativo.
- Linha 6 — É nessa linha que acontece a leitura dos dados no banco, pelo comando `mysql_query($sql)`. Caso ocorra algum erro, ele será informado.

- Linhas 7 a 13 — Enquanto for possível realizar a leitura dos registros na tabela, ela exibirá (retornará para o aplicativo) o conteúdo dos campos `Nome`, `Local`, `Distancia`, `Data` e `Largada`, seguido de um símbolo `#` após cada item da corrida selecionada. Esse símbolo, como já dissemos, será usado para separar cada um dos campos da corrida.

- Linha 14 — Quando acabarem os campos cadastrados na tabela, vamos inserir um símbolo (`^`) para indicar ao App Inventor que os registros terminaram.

- Linha 15 — Encerra o bloco da estrutura de comandos do PHP.

Assim como os demais arquivos em PHP, esse deverá ser gravado na mesma pasta, ou seja, dentro da `app_run` que se encontra no diretório `root` do servidor USBWebserver.

De volta ao ambiente do App Inventor, necessitamos programar os blocos da ação do usuário **após a seleção** do nome da corrida na lista que foi exibida. Para isso, localize o componente `List_Corrida` e selecione a opção `when List_Corrida.AfterPicking` para inserir na área de `Viewer`.

Figura 10.10: Bloco List_Corrida.AfterPicking

Assim como configuramos os outros componentes `Web`,

precisamos também configurar o `Web_Selecionar` para indicar o caminho do arquivo `selecionar.php`. Insira o `set Web_Selecionar.Url to` e encaixe um bloco de texto com as informações: http://seunumeroIP:8080/app_run/selecionar.php.

> A cada dia que o leitor for testar seu aplicativo, antes de emular, é preciso verificar o número de IP, pois ele pode ser alterado. Caso tenha sido modificado, altere as configurações dos componentes `Web` de todo o projeto. Essas verificações terminarão quando colocarmos o banco de dados em um servidor online.

Agora, precisamos enviar a informação do nome da corrida para que o arquivo `selecionar.php` realize a consulta no banco de dados e retorne-a para o aplicativo. Insira um bloco `call Web_Selecionar.PostText`. Teremos de enviar a variável `nome=` com o nome da corrida que foi selecionada e está representada no bloco de comando `List_Corrida.Selection`.

Para isso, insira um bloco de texto `join` na opção `text` do bloco `call Web_Selecionar.PostText`. Nele encaixe um novo bloco de texto com o conteúdo `nome=`. Para indicar o nome da corrida selecionada, encaixe o `List_Corrida.Selection`. Dessa maneira, a informação com esse nome será enviada para o arquivo `selecionar.php` e este realizará uma consulta no banco de dados. Veja o resultado do bloco `when List_Corrida.AfterPicking` na figura a seguir.

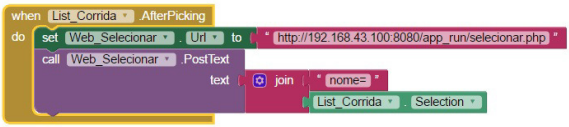
when List_Corrida .AfterPicking
do set Web_Selecionar . Url to " http://192.168.43.100:8080/app_run/selecionar.php "
call Web_Selecionar .PostText
text join " nome= "
List_Corrida . Selection

Figura 10.11: when List_Corrida.AfterPicking

Após o arquivo `selecionar.php` ser acionado pelo componente `call Web_Selecionar.PostText`, as informações da corrida serão retornadas ao aplicativo. Para receber e tratarmos essas informações, precisamos de um bloco que realize essa tarefa. Insira um bloco `when Web_Selecionar.GotText` para receber esses dados e trabalhá-los para exibição na tela.

Figura 10.12: when Web_Selecionar.GotText

Dentro desse bloco de controle, teremos três decisões:

1. Verificaremos se o símbolo retornado é o `^`, pois ele nos indica que nenhuma corrida foi encontrada.

2. Se o texto recebido possuir o caractere `#`, deverá ser tratado para a exibição.

3. Se não for nenhuma das opções anteriores, isso nos indica que um erro ocorreu no processamento.

Vamos inserir um bloco de controle `if` e configurá-lo para possuir a opção `else if` e também um `else`. Veja como deverá

ficar esse bloco de controle:

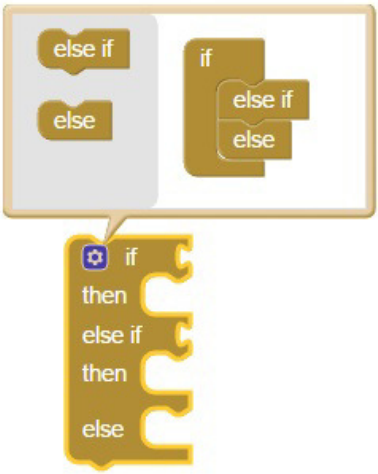

Figura 10.13: Bloco if configurado

Assim como verificamos a existência de registros no banco de dados para o `listar.php` com o símbolo `^`, realizaremos a mesma tarefa para o arquivo `selecionar.php`.

Vamos preparar a condição de decisão que teremos no comando `if`. Como é preciso verificar se o texto recebido do arquivo `selecionar.php` é igual ao caractere que indica o seu final, o `^`, encaixe um bloco de comparação com o símbolo de `=`, da guia `Built-in` na seção `Logic`, no bloco `if`.

No primeiro espaço livre, precisamos encaixar o bloco que recebe as informações vindas do arquivo `selecionar.php`. Selecione o `get responseContent` conforme as instruções vistas anteriormente.

Para verificar se a informação recebida foi o `^`, temos de

colocar um bloco de texto e digitar esse símbolo. A comparação verifica se recebemos **apenas** esse símbolo; se isso ocorrer, não temos nenhuma corrida cadastrada e deveremos exibir essa informação ao usuário. Para isso, após o `then`, insira um bloco `call Notifier1.ShowAlert` e nele encaixe um bloco de texto na opção `notice` com a informação: **Nenhuma corrida encontrada**. Veja o resultado:

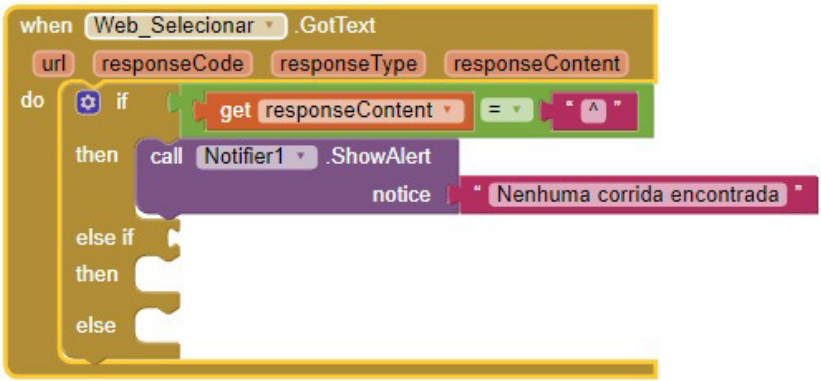

Figura 10.14: Configuração do if

Na segunda decisão `else if`, necessitamos verificar se o texto recebido do arquivo `selecionar.php` contém o caractere `#`, para então tratarmos os valores. Insira o bloco de texto que realiza essa verificação, o `contains text ... piece`, na frente da opção `else if`. Em frente ao `containts text`, encaixe o bloco `get responseContent`, que recebe as informações do arquivo `selecionar.php` e na opção `Web_Selecionar.GotText`.

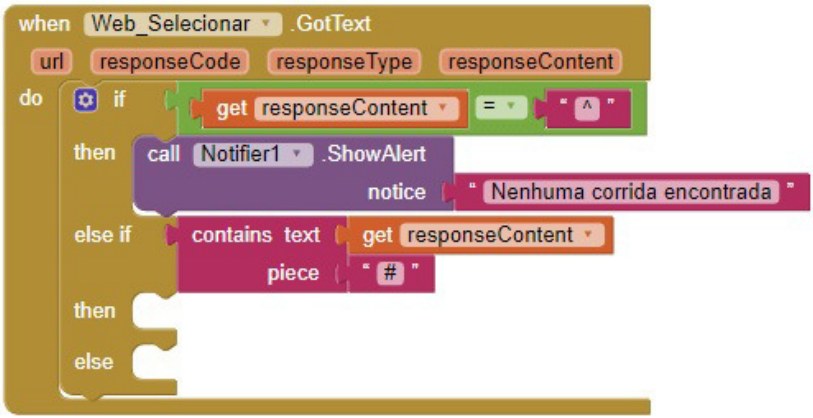
when Web_Selecionar .GotText
url responseCode responseType responseContent
do if get responseContent = " ^ "
then call Notifier1 .ShowAlert
notice " Nenhuma corrida encontrada "
else if contains text get responseContent
piece " # "
then
else

Figura 10.15: Web_Selecionar.GotText

Precisamos novamente realizar a substituição do caractere `#` pela vírgula (`,`) e armazenar todo o resultado obtido pela troca na variável `separa` . Logo, coloque o bloco da variável `set global separa to` e encaixe-o logo após a opção `then` do `else if` .

Para realizar a substituição dos símbolos, usaremos a mesma lógica vista anteriormente para listar todos os nomes das corridas, ou seja, teremos de trocar todos os símbolos de `#` pela vírgula. A próxima figura demonstra como deverá ficar o bloco finalizado:

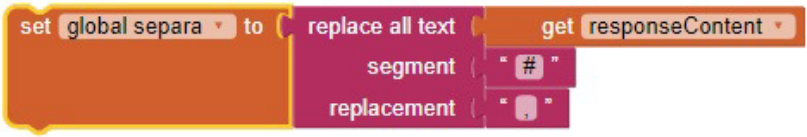

Figura 10.16: Substituição de caracteres

Como visto anteriormente, ainda temos o sinal que indica o fim dos registros, o acento circunflexo (`^`). Utilizando a mesma

técnica já aprendida, faça a substituição desse símbolo por um espaço vazio. A figura exibe o bloco que retira a marca de final do registro.

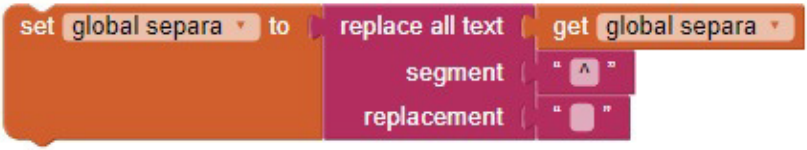

Figura 10.17: Retirando marca de final

As informações da corrida, separadas pela vírgula e sem a marcação de final do arquivo, já estão armazenadas na variável `separa` e devem ser colocadas no `List_Selecionar` para depois serem posicionadas nas caixas de texto da tela do aplicativo. Então, vamos adicionar o componente `set List_Selecionar.ElementsFromString to` para receber os dados da variável `separa` .

Depois, vamos adicionar o bloco da variável `get global separa` . A figura a seguir exibe o bloco que insere os dados da corrida da variável para o `List_Selecionar` .

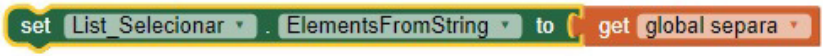

Figura 10.18: Bloco List_Selecionar

Quando atribuímos a variável `separa` ao `List_Selecionar` , foi criada uma lista na memória do dispositivo, contendo uma informação da corrida em cada linha. A primeira linha contém o nome da **corrida**; a segunda contém o **local**; a terceira tem a **distância**; a quarta, a **data**; e a quinta possui o **horário**. A ordem

em que as informações aparecem é a mesma que está na tabela do banco de dados. Conseguimos identificar cada linha do `List_Selecionar`, utilizando a propriedade `SelectionIndex` e dizendo o número da linha que desejamos.

Vamos posicionar a `List_Selecionar` na primeira linha para podermos selecionar o nome da corrida para posterior exibição em tela. Insira um bloco `set List_Selecionar.SelectionIndex to` e um bloco de número da guia `Built-in` da seção `Math`. Dentro dele, digite o número da linha desejada, ou seja, `1`.

Já que estamos identificando o nome da corrida, vamos armazená-la também para um futuro processo de alteração dos seus dados, que será visto nos próximos capítulos. Para isso, devemos criar uma variável com o nome `nomevelho` e encaixar um bloco de texto vazio.

Insira um `set global nomevelho to` e, na sequência, para guardar o nome atual, coloque o bloco que contém a linha com essa informação. Vá até a guia de blocos e localize o `List_Selecionar.Selection`. Para exibir o nome da corrida, insira um `set Txt_Corrida.Text to`, e depois posicione o bloco do componente `List_Selecionar.Selection`. A figura mostra o procedimento para exibir o nome da corrida na tela do aplicativo:

set List_Selecionar . SelectionIndex to 1
set global nomevelho to List_Selecionar . Selection
set Txt_Corrida . Text to List_Selecionar . Selection

Figura 10.19: Exibindo o nome da corrida

Teremos de repetir esse processo para as demais `TextBoxs`.

Veja na tabela os blocos que deveremos programar:

Componente que receberá a informação	Bloco com a informação
List_Selecionar to	2
Txt_Local.Text	List_Selecionar.Selection
List_Selecionar to	3
Txt_Distancia.Text	List_Selecionar.Selection
List_Selecionar to	4
Txt_Data.Text	List_Selecionar.Selection
List_Selecionar to	5
Txt_Horario.Text	List_Selecionar.Selection

A figura a seguir apresenta os blocos citados para a exibição dos valores na tela.

set List_Selecionar . SelectionIndex to 2
set Txt_Local . Text to List_Selecionar . Selection
set List_Selecionar . SelectionIndex to 3
set Txt_Distancia . Text to List_Selecionar . Selection
set List_Selecionar . SelectionIndex to 4
set Txt_Data . Text to List_Selecionar . Selection
set List_Selecionar . SelectionIndex to 5
set Txt_Horario . Text to List_Selecionar . Selection

Figura 10.20: Exibindo os campos da corrida

A terceira decisão que teremos de tratar é a que verifica se "ocorreu um erro" em alguma das decisões anteriores. Como só está faltando a opção do `else` , é nela que vamos encaixar o bloco que exibirá a mensagem de erro.

Insira um bloco `call Notifier1.ShowAlert` e nele encaixe um bloco de texto na opção `notice`, com a informação: **Ocorreu um erro**. A seguir, veja o bloco completo do `Web_Selecionar.GotText`:

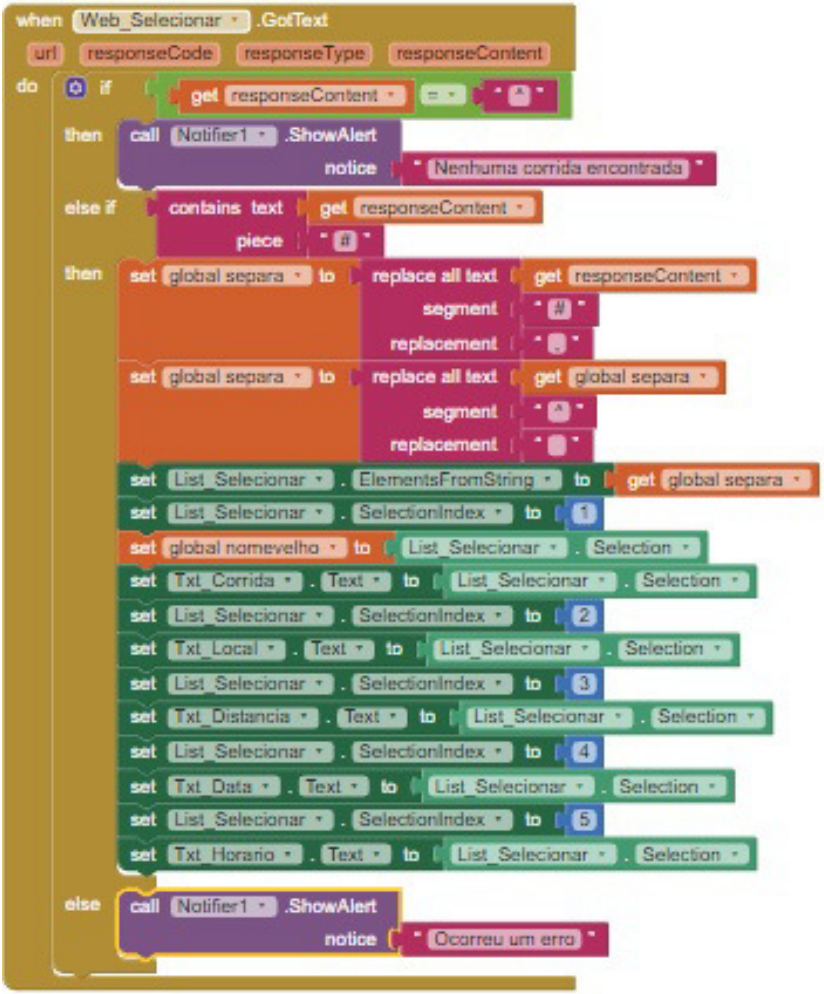
when Web_Selecionar .GotText
url responseCode responseType responseContent
do if get responseContent = " "
then call Notifier1 .ShowAlert
notice Nenhuma corrida encontrada
else if contains text get responseContent
piece #
then set global separa to replace all text get responseContent
segment #
replacement ,
set global separa to replace all text get global separa
segment "
replacement
set List_Selecionar . ElementsFromString to get global separa
set List_Selecionar . SelectionIndex to 1
set global nomevelho to List_Selecionar . Selection
set Txt_Corrida . Text to List_Selecionar . Selection
set List_Selecionar . SelectionIndex to 2
set Txt_Local . Text to List_Selecionar . Selection
set List_Selecionar . SelectionIndex to 3
set Txt_Distancia . Text to List_Selecionar . Selection
set List_Selecionar . SelectionIndex to 4
set Txt_Data . Text to List_Selecionar . Selection
set List_Selecionar . SelectionIndex to 5
set Txt_Horario . Text to List_Selecionar . Selection
else call Notifier1 .ShowAlert
notice Ocorreu um erro

Figura 10.21: Bloco Web_Selecionar.GotText

10.6 EMULANDO A CONSULTA

Vamos emular nosso aplicativo no próprio ambiente do App Inventor. Aguarde o seu carregamento por completo no emulador e clique sobre o botão `Listar Corridas` para exibir a lista, conforme visto no começo deste capítulo. Veja como fica:

Figura 10.22: Listando as corridas

Ao clicar em uma das opções mostradas na lista, todas as

informações referentes a essa corrida deverão ser exibidas na tela de seu aplicativo:

Figura 10.23: Exibindo todos os dados da corrida

Caso ocorra algum erro, reveja todos os procedimentos adotados neste capítulo, principalmente se os comandos digitados nos arquivos em PHP estão exatamente como demonstrados e se respeitam as letras maiúsculas e minúsculas. Veja também o número do IP que está utilizando e se todos os blocos estão

conforme feitos aqui.

10.7 RESUMINDO

Neste capítulo, vimos duas maneiras de realizar a consulta das informações que estavam no banco de dados: uma para exibir na `ListPicker` em formato de lista, e a outra para mostrar todos os detalhes da corrida nas `TextBoxs` .

No próximo capítulo, adicionaremos o botão para realizar a exclusão de uma corrida que está armazenada no banco de dados.

Capítulo 11

EXCLUINDO UM REGISTRO

Após a inclusão, listagem e exibição dos dados da corrida selecionada, vamos dar sequência ao projeto adicionando a função de excluir um registro. Precisaremos adicionar mais um botão para realizar a exclusão, e também mais um componente `Web` para enviar a solicitação de exclusão ao arquivo em PHP, que veremos mais à frente.

A tabela a seguir demonstra os novos componentes que devemos inserir e suas propriedades para alteração. Comece inserindo um novo `HorizontalArrangement` logo abaixo dos botões `Salvar` e `Listar Corridas`.

Componente	Guia	Propriedade	Alterar valor para
HorizontalArrangement	Layout	AlignHorizontal	Center
		AlignVertical	Center
		Width	Fill Parent
Button	User Interface	Bold	Marcar
		FontSize	16
		FontColor	red

		Text	Excluir
		Rename	Btn_Excluir
Web	Connectivity	Rename	Web_Excluir

Para o componente `HorizontalArrangement` , deixamos o alinhamento referente às partes **Horizontal** e **Vertical** marcado como `Center` , para que o próximo componente inserido em seu interior fique centralizado. Deixamos também a propriedade do comprimento como `Fill Parent` .

Para o `Button` , realizamos as alterações para o efeito de negrito, aumentamos o tamanho da fonte, trocamos sua cor para `red` e digitamos `Excluir` no `text` . Alteramos o nome do componente para `Btn_Excluir` .

Inserimos aqui outro componente (`Web`) que realiza o envio de dados para um arquivo em PHP, e alteramos seu nome para melhor identificação durante a programação dos blocos. Veja como ficará a visualização da tela do aplicativo:

Figura 11.1: Botão Excluir

11.1 ARQUIVO EXCLUIR.PHP

Conforme vimos nos capítulos anteriores, aqui também necessitamos enviar a solicitação de exclusão para um arquivo na linguagem PHP realizar a tarefa. A seguir, veja as linhas de código do arquivo `excluir.php` :

```
1. <?php
2.   $nome=$_POST['nome'];
3.   $conexao = mysql_connect('localhost','root','usbw');
4.   mysql_select_db('CORRIDA',$conexao);
5.   $sql = "delete from AGENDA where Nome like '$nome' ";
6.   $resultado = mysql_query($sql) or die ("Erro: " . mysql_erro
r());
7.   if($resultado)
8.        echo "1";
9.    else
10.       echo "0";
11. ?>
```

Vamos entender os códigos desse arquivo que realiza a exclusão de uma corrida cadastrada no banco de dados:

- Linha 1 — Tag que indica a abertura de um arquivo em PHP.
- Linha 2 — Variável `$nome` que recebe o nome da corrida enviada pelo aplicativo através do método `POST` .
- Linha 3 — Estamos criando a variável `$conexao` que realizará uma conexão do arquivo PHP com o phpMyAdmin. Nessa linha, são informados o localhost, o nome do usuário e a senha de acesso ao gerenciador de banco de dados, como já vimos anteriormente.
- Linha 4 — Realiza a identificação do banco de dados que vamos utilizar no projeto, ou seja, o banco `CORRIDA` .
- Linha 5 — Para a variável `$sql` , definimos a instrução em

SQL para deletar os dados no banco de dados referente ao nome da corrida selecionada pelo aplicativo.

- Linha 6 — Realiza efetivamente a exclusão dos dados no banco e, caso ocorra um erro, ele será enviado para o aplicativo.
- Linhas 7 a 10 — Pelo comando de decisão `if` , verifica se os comandos da linha 6 foram concluídos com êxito. Caso as informações tenham sido excluídas no banco, será exibido o número 1; caso ocorra algum problema com a exclusão dos dados, será exibido o número 0.
- Linha 11 — Encerra o bloco da estrutura de comandos do PHP.

Assim como os outros arquivos em PHP usados no projeto, esse também deverá ficar gravado na mesma pasta que os demais, ou seja, `app_run` .

11.2 BOTÃO EXCLUIR

Voltando para o App Inventor, vamos começar a programar o bloco que realiza a exclusão do registro de uma corrida. Insira um bloco `when Btn_Excluir.Click` na sua área de blocos. A partir de sua ativação pelo clique do usuário, deveremos primeiramente verificar se existe na tela do aplicativo alguma corrida para ser excluída. Se não existir, deveremos avisar o usuário.

Para essa verificação, insira um bloco de controle `if` e configure a opção `else` . Para verificar se o nome da corrida que está no `Txt_Corrida.Text` está vazio, insira um bloco de texto

`is empty`, e depois encaixe o bloco que deverá ser testado, o `Txt_Corrida.Text`.

Caso o bloco `Txt_Corrida.Text` esteja sem nenhum valor, insira um bloco do componente `call notifier1.ShowAlert` para dar um aviso ao usuário. Encaixe o bloco de texto e digite em seu interior a informação: **Nenhuma corrida foi selecionada**. A figura a seguir exibe o bloco com a verificação da existência de um nome no componente `Txt_Corrida.Text`.

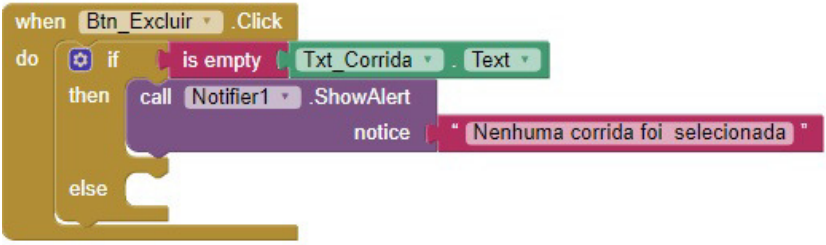
when Btn_Excluir .Click
do if is empty Txt_Corrida . Text
then call Notifier1 .ShowAlert
notice " Nenhuma corrida foi selecionada "
else

Figura 11.2: Verificação do texto

No caso da existência de uma corrida na tela, devemos configurar o componente `Web_Excluir` na opção `else` para localizar o endereço do arquivo `excluir.php`. Para isso, insira um bloco do componente `set Web_Excluir.Url to` e nele encaixe um bloco de texto com o local onde se encontra o arquivo de exclusão. Você deverá digitar: http://seunumeroIP:8080/app_run/excluir.php.

Para realizar o envio dos dados para o arquivo PHP, devemos utilizar um bloco do componente `call Web_Excluir.PostText` e nele encaixar o nome da corrida para a exclusão. Complete com um bloco `Join`, informe o nome da variável que será enviada `nome=` e complete com o bloco do componente

`Txt_Corrida.Text` para o envio da solicitação. Veja na figura a seguir o bloco completo:

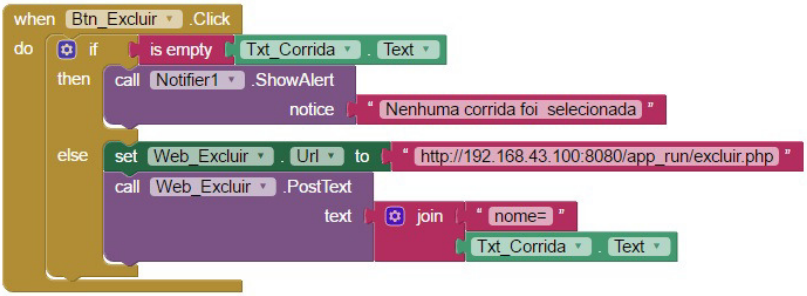
when Btn_Excluir .Click
do if is empty Txt_Corrida . Text
then call Notifier1 .ShowAlert
notice " Nenhuma corrida foi selecionada "
else set Web_Excluir . Url to " http://192.168.43.100:8080/app_run/excluir.php "
call Web_Excluir .PostText
text join " nome= "
Txt_Corrida . Text

Figura 11.3: Btn_Excluir

Depois do envio dos dados e do processamento da exclusão por parte do arquivo `excluir.php`, devemos verificar o que foi retornado de informação para o nosso app e, a partir disso, informar ao usuário o que ocorreu. Se a informação retornada for o número **1**, o registro foi apagado; se for o número **0**, um erro ocorreu durante o processo e essa informação deverá aparecer para o usuário.

Para tanto, insira um bloco `Web_Excluir.GotText` para receber a informação de retorno do arquivo `excluir.php`. Dentro dele, necessitamos de um bloco de controle `if` configurado com um `else`. Assim como fizemos no botão para incluir uma corrida, aqui também precisamos verificar o conteúdo do texto retornado.

Encaixe no `if` um bloco de texto `contains text`. Na primeira opção, encaixe o bloco que identifica o valor retornado, o `responseContent`. Na opção `piece`, inclua um bloco de texto com o número `1` dentro dele.

Se o resultado dessa verificação for verdadeiro, devemos exibir uma mensagem ao usuário. Insira um bloco `call notifier1.ShowAlert` e nele um bloco de texto com o valor **Corrida EXCLUÍDA**. Caso o resultado seja diferente de **1**, vamos inserir outro bloco `call notifier1.ShowAlert` após o comando `else`, e configurá-lo para a exibição da mensagem: **Não foi possível EXCLUIR**.

Logo após o bloco do comando `if`, é preciso limpar todas as informações que estão visíveis na tela do seu aplicativo. Insira os blocos de texto de cada campo e neles encaixe um bloco de texto vazio. Veja a seguir o bloco completo do `Web_Excluir.GotText`.

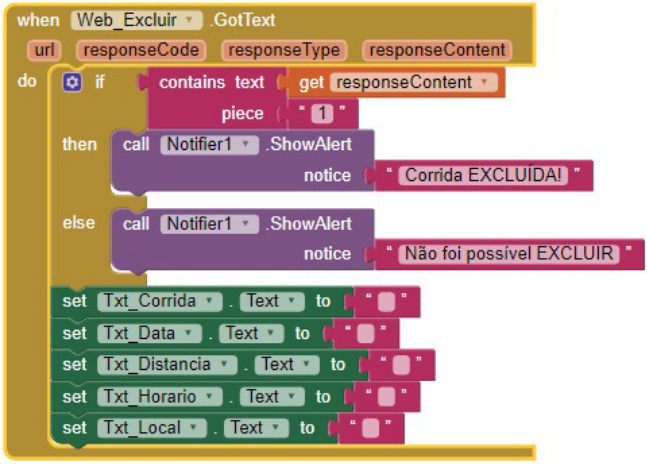
when Web_Excluir .GotText
url responseCode responseType responseContent
do if contains text get responseContent
piece " 1 "
then call Notifier1 .ShowAlert
notice " Corrida EXCLUIDA! "
else call Notifier1 .ShowAlert
notice " Não foi possível EXCLUIR "
set Txt_Corrida . Text to " "
set Txt_Data . Text to " "
set Txt_Distancia . Text to " "
set Txt_Horario . Text to " "
set Txt_Local . Text to " "

Figura 11.4: Web_Excluir.GotText

11.3 EMULANDO A EXCLUSÃO

Chegamos ao momento de testar o botão de excluir. Emule novamente pelo ambiente de desenvolvimento. Realize uma consulta para que os dados sejam exibidos na tela do aplicativo,

conforme:

Figura 11.5: Testando o app

Após o clique no botão de excluir, você deverá receber a mensagem de **Corrida EXCLUÍDA**.

Figura 11.6: Confirmação de exclusão

Caso a mensagem **Não foi possível EXCLUIR** seja exibida, teremos de rever os procedimentos adotados neste capítulo. Reveja se o arquivo `excluir.php` está exatamente como o demonstrado aqui. Vale lembrar que devemos respeitar as letras que estão em minúsculas e maiúsculas, pois o PHP é *case-sensitive*. Também verifique o número do IP de seu computador e os códigos que foram digitados dentro dos blocos.

11.4 RESUMINDO

No próximo capítulo, concluiremos o aplicativo de corridas implementando a função do botão de alterar alguma informação da corrida já cadastrada.

CAPÍTULO 12

ALTERANDO UM REGISTRO

Neste capítulo, finalizaremos nosso aplicativo de controle de corridas para conseguirmos realizar as quatro operações básicas com um banco de dados. Já aprendemos a **inserir**, **consultar** e **excluir** dados, e agora veremos o botão de **alterar** um registro já existente.

Vamos aprender como modificar dados recuperados da tabela e salvar as atualizações realizadas. Ao final, deixaremos seu aplicativo online e pronto para ser usado no seu dispositivo móvel.

12.1 FINALIZANDO O LAYOUT

Para completar a tela do aplicativo, precisamos inserir mais dois componentes em nossa tela de design: um botão para solicitar a alteração e um componente `Web` para enviar as informações para o arquivo em PHP processar a solicitação. Veja na tabela a seguir os novos componentes inseridos e suas propriedades. Insira o novo `Button` ao lado do botão `Excluir` .

Componente	Guia	Propriedade	Alterar valor para
Button	User Interface	Bold	Marcar

		FontSize	16
		FontColor	red
		Text	Alterar
		Rename	Btn_Alterar
Web	Connectivity	Rename	Web_Alterar

Para o componente `Button`, realizamos as alterações para o efeito de negrito, aumentamos o tamanho da fonte, trocamos sua cor para `red` e digitamos `Alterar` para o `text`. Também alteramos o nome do componente para `Btn_Alterar`.

Inserimos mais um componente (`Web`), que realiza o envio de dados para um arquivo em PHP, e alteramos seu nome para uma melhor identificação durante a programação dos blocos. Observe como ficará a visualização da tela finalizada do aplicativo:

Figura 12.1: Tela completa do app

12.2 ARQUIVO ALTERAR.PHP

Novamente, antes de abrirmos a seção de blocos para a programação, vamos criar o arquivo `alterar.php` que executará a solicitação de alteração do registro, quando o usuário clicar no

botão `Alterar`. A seguir, veja as suas linhas de código:

```
1.  <?php
2.    $nomenovo=$_POST['nomenovo'];
3.    $nomevelho=$_POST['nomevelho'];
4.    $local=$_POST['local'];
5.    $distancia=$_POST['distancia'];
6.    $data= date('Y-d-m', strtotime($_POST['data']));
7.    $largada=$_POST['largada'];
8.    $conexao = mysql_connect('localhost','root','usbw');
9.    mysql_select_db('CORRIDA',$conexao);
10.   $sql = "update AGENDA set Nome='$nomenovo', Local='$local',
 Distancia='$distancia', Data='$data', Largada='$largada' where N
ome='$nomevelho'";
11.   $resultado = mysql_query($sql) or die ("Erro .:" . mysql_er
ror());
12.   if($resultado)
13.          echo "1";
14.   else
15.       echo "0";
16.  ?>
```

Vamos entender as linhas internas do arquivo `alterar.php`:

- Linha 1 — Tag que indica a abertura de um arquivo em PHP.
- Linhas 2 a 7 — Através do método `$_POST`, recebemos o valor passado pelo aplicativo e atribuímos suas informações às variáveis locais dentro do arquivo PHP.
- Linha 8 — Criamos a `$conexao` com o phpMyAdmin, na qual são informados o localhost, o usuário e a senha de acesso ao gerenciador de banco de dados.
- Linha 9 — Realiza a identificação do banco de dados que vamos utilizar no projeto, o `CORRIDA`.
- Linha 10 — Para a variável `$sql`, estamos definindo a

instrução em SQL para alterar os dados no banco de dados referente ao nome da corrida selecionada pelo aplicativo, onde passamos todos os valores necessários para a alteração das informações.

- Linha 11 — Realiza efetivamente a alteração dos dados no banco e, caso ocorra um erro, ele será exibido.
- Linhas 12 a 15 — Pelo comando de decisão `if` , verifica se os comandos da linha 11 foram concluídos com êxito. Caso as informações tenham sido alteradas no banco, será exibido o número **1**; caso ocorra algum problema, será exibido o número 0.
- Linha 16 — Encerra o bloco da estrutura de comandos do PHP.

Não se esqueça de salvar esse arquivo com os demais arquivos em PHP. Vale lembrar que todos deverão ficar armazenados na pasta `app_run` .

12.3 A LÓGICA DO BOTÃO ALTERAR

Voltando para o ambiente de desenvolvimento do App Inventor, clique no botão `Blocks` para podermos realizar as tarefas para alterar um registro que já se encontra visualizado em tela. Vejamos a seguir a lógica que usaremos para ele.

1. Vamos verificar se existe uma corrida exibida no bloco da variável `nomevelho` , pois, se existir, é porque uma já foi selecionada através do botão `Listar Corridas` e ela está visível na tela do aplicativo. No caso da variável estar sem

nenhum valor, exibiremos uma mensagem ao usuário.

2. No caso da existência de um valor na variável `nomevelho`, vamos configurar o endereço do componente `Web_Alterar` para encontrar o arquivo que realiza essa função de alteração.

3. Através do método `Post` do componente `Web_Alterar`, definiremos os campos que serão enviados para o arquivo em PHP.

4. Limparemos todas as informações que estão disponíveis na tela após a sua alteração no banco de dados.

5. Depois do envio dos dados, verificaremos qual o resultado obtido pelo evento `GotText`, que recebe as informações de retorno do arquivo em PHP, para então avisar ao usuário se a solicitação foi concluída com sucesso ou não.

12.4 PROGRAMANDO A ALTERAÇÃO

Com base no primeiro passo da lógica vista, vamos inserir um bloco e ativar a solicitação para alterar um registro. Insira um bloco `when Btn_Alterar.Click` na sua área de `Viewer`. Dentro dele, para realizar a verificação da existência de um valor na variável `nomevelho`, insira um bloco de controle `if` e configure a opção `else`.

Na sequência desse bloco, coloque o bloco lógico de comparação (`=`) e, no primeiro espaço disponível, insira `get global nomevelho` e, no outro espaço, complete com um bloco de texto vazio. Após a opção `then`, insira um bloco `call`

`Notifier1.ShowAlert` e configure-o para exibir a informação: **Selecione uma corrida**.

Veja na figura a seguir como ficará a parte inicial do `Btn_Alterar`:

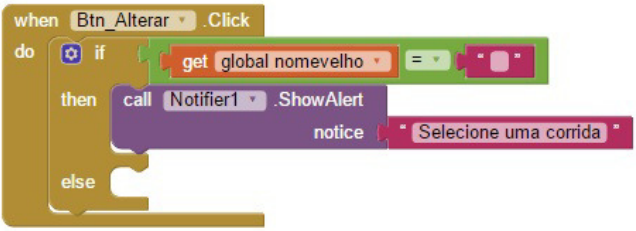

Figura 12.2: Btn_Alterar

Se existir um nome de corrida na tela, devemos configurar o endereço do componente `Web_Alterar` logo a seguir da opção `else`, para localizar o arquivo `alterar.php`. Para isso, insira um bloco do componente `set Web_Alterar.Url to` e nele encaixe um bloco de texto com o local onde se encontra o arquivo desejado. Você deverá digitar: http://seunumeroIP:8080/app_run/alterar.php.

Após indicar o local do arquivo `alterar.php`, teremos de informar os dados que serão enviados para o arquivo em PHP. Como vimos anteriormente, insira o bloco do componente que realiza o envio: `call Web_Alterar.PostText`. Como teremos várias informações para enviar, necessitamos de um bloco de texto `join`. Além das duas opções de encaixe já apresentadas, configure-o para receber mais 10 opções de encaixe.

Para cada opção de encaixe, vamos precisar de um bloco de texto representando uma variável que será enviada para o arquivo

`alterar.php` e, no encaixe imediatamente abaixo, um bloco do componente que contém essa informação. A tabela a seguir exibe a ordem e os valores que devem ser encaixados nas opções.

Informação	Componente
nomenovo=	Bloco de texto
Txt_Corrida.Text	Bloco de componente
&nomevelho=	Bloco de texto
get globalnomevelho	Bloco de componente
&local=	Bloco de texto
Txt_Local.Text	Bloco de componente
&distancia=	Bloco de texto
Txt_Distancia.Text	Bloco de componente
&data=	Bloco de texto
Txt_Data.text	Bloco de componente
&largada=	Bloco de texto
Txt_Horario.Text	Bloco de componente

O resultado da junção dos comandos utilizados no bloco `join` produzirá a seguinte linha de comando:

```
nomenovo=Txt_Corrida.Text&nomevelho=nomevelho&local=Txt_Local.Tex
t&distancia=Txt_Distancia.Text&data=Txt_Data.text&largada=Txt_Hor
ario.Text
```

Ela será enviada para o arquivo `alterar.php` processar a alteração no banco de dados. Veja na figura a seguir como ficarão os blocos do componente `Web_Alterar` .

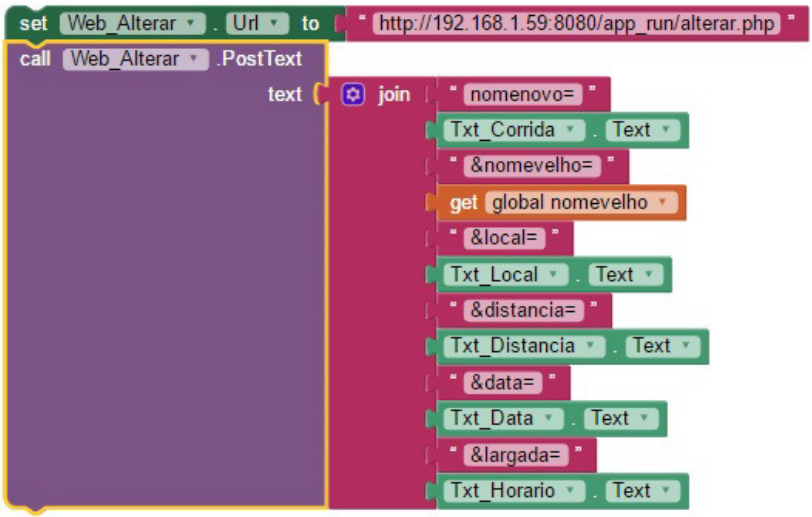

Figura 12.3: Web_Alterar

Após o envio das informações, precisamos limpar os dados que já foram usados e estão visíveis em sua tela, além de apagar o valor da variável `nomevelho`. Para isso, vamos inserir um bloco de texto vazio nos blocos dos componentes que representam essas informações: `set Txt_Corrida.Text to`, `set Txt_Data.Text to`, `set Txt_Distancia.Text to`, `set Txt_Horario.Text to`, `set Txt_Local.Text to` e `set global nomevelho to`.

Vamos também solicitar que o aplicativo recolha o teclado virtual que foi utilizado para a digitação das informações. Localize o bloco do `call Txt_Local.HideKeyboard` e adicione-o logo após os componentes que estão limpando as informações.

set Txt_Corrida . Text to ""
set global nomevelho to ""
set Txt_Data . Text to ""
set Txt_Distancia . Text to ""
set Txt_Horario . Text to ""
set Txt_Local . Text to ""
call Txt_Local .HideKeyboard

Figura 12.4: Limpando as informações e recolhendo o teclado

Veja na próxima figura como ficará o bloco do botão `Btn_Alterar.Click` finalizado:

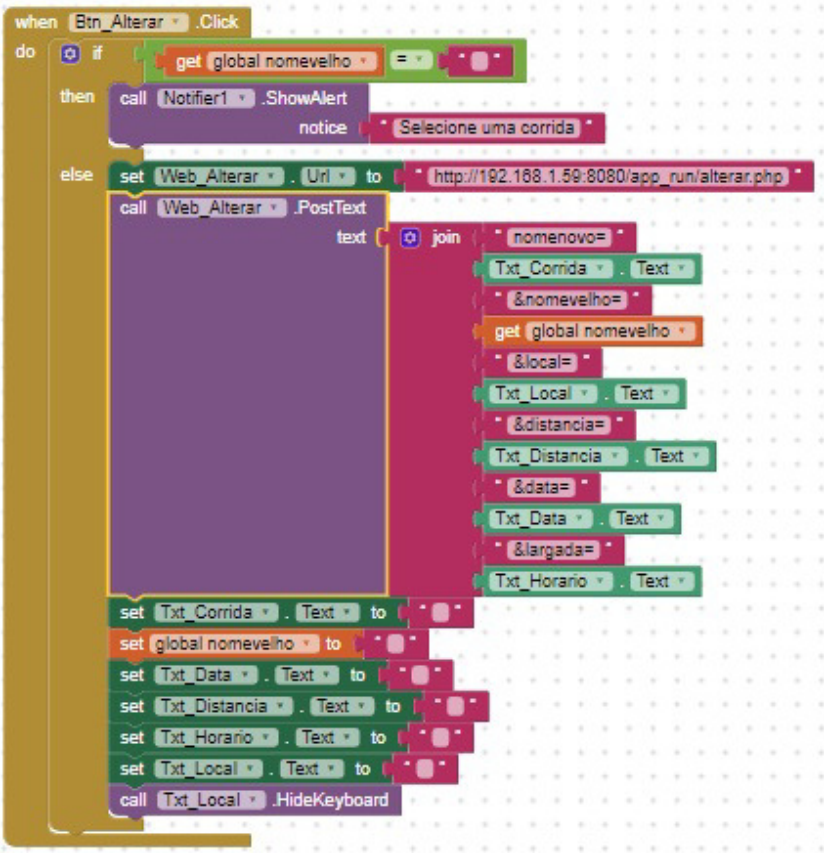

Figura 12.5: Botão Alterar

Depois do envio dos dados e do processamento das alterações por parte do arquivo `alterar.php`, devemos verificar a informação que foi retornada para o nosso app e, a partir dela, informar ao usuário o que ocorreu. Se a informação retornada for o número **1**, o registro foi alterado com sucesso, mas se for o número **0**, ocorreu um erro durante o processo e essa informação também deverá aparecer para o usuário.

Para tanto, insira um bloco `Web_Alterar.GotText` para receber a informação de retorno do arquivo `alterar.php`. Dentro dele, necessitamos de um bloco de controle `if` configurado com um `else`. Assim como fizemos para os outros botões, também precisamos verificar o conteúdo do texto retornado.

Encaixe no `if` um bloco de texto `contains text`. Na primeira opção, encaixe o bloco que identifica o valor retornado, o `responseContent`. Na opção `piece`, inclua um bloco de texto com o número `1` dentro dele.

Se o resultado dessa verificação for verdadeiro, exibiremos a mensagem de que a alteração foi realizada com sucesso. Insira um bloco `call notifier1.ShowAlert` e nele um bloco de texto com o valor: **Corrida atualizada!**. Caso o resultado seja diferente de **1**, após o comando `else`, vamos colocar outro bloco `call notifier1.ShowAlert` e configurá-lo para a exibição da mensagem: **Não foi possível atualizar**. A figura a seguir demonstra o bloco `Web_Alterar.GotText` finalizado.

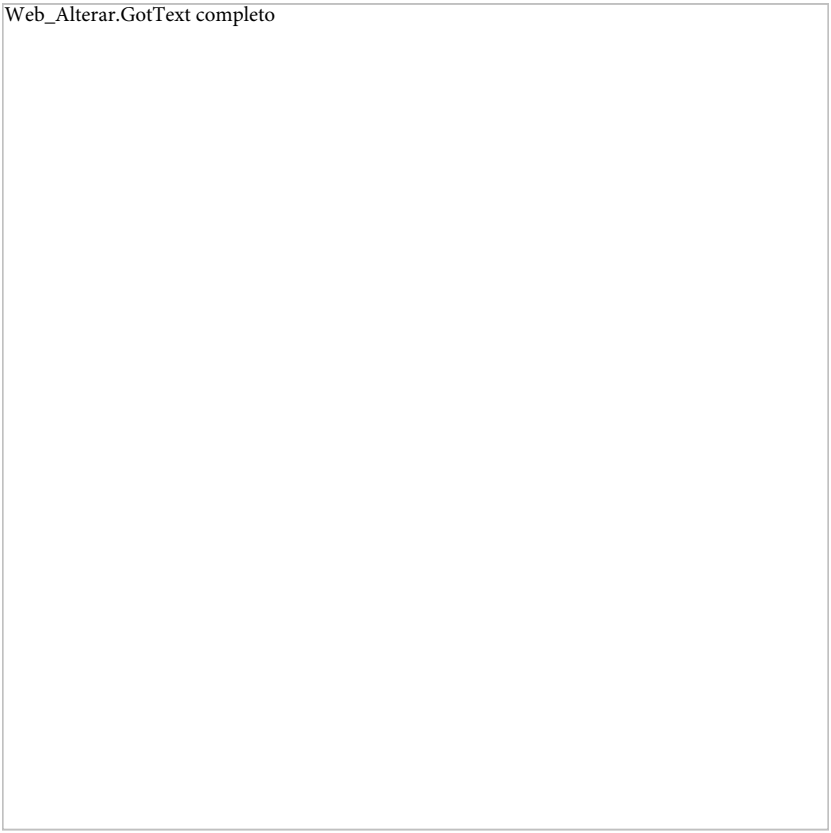

Figura 12.6: Web_Alterar.GotText completo

12.5 EMULANDO A ALTERAÇÃO

Teste a função de alteração dos dados, emulando diretamente no seu computador. Após realizar uma consulta de uma corrida qualquer, digite as informações que você deseja alterar na `TextBox` específica e clique no botão `Alterar` . Caso a alteração tenha ocorrido com sucesso, teremos:

Figura 12.7: Alteração concluída com sucesso

Caso ocorra algum erro, reveja todos os procedimentos adotados neste capítulo, principalmente se os comandos digitados nos arquivos estão exatamente como demonstrados, respeitando o *case-sensitive*. Veja também o número do IP que está utilizando e se todos os blocos estão conforme expostos anteriormente.

12.6 CONECTANDO COM UM BANCO DE DADOS ONLINE

Como vimos até agora, em todos os acessos ao banco de dados, necessitamos identificar o número do IP para que nosso aplicativo encontrasse o endereço no servidor web local. Essa prática foi adotada por questões meramente didáticas, porém, a partir de agora, vamos deixar nosso aplicativo funcionando com o banco de dados instalado em um provedor hospedeiro.

> O leitor necessitará de um provedor hospedeiro que disponibilize o acesso remoto ao MySQL. Geralmente, esse serviço tem um custo, mas existem bons servidores com um preço bem camarada para a hospedagem. Eu utilizo e recomendo o **HOSTINGER**, disponível em: http://www.hostinger.com.br.

Você deverá ter uma conta em um provedor hospedeiro qualquer, mas verifique se ele disponibiliza o recurso do **acesso remoto** a seu banco de dados. A criação de um banco de dados em um provedor online é realizada através do ambiente do phpMyAdmin, da mesma maneira como aprendemos anteriormente; apenas ele estará online.

Após a criação do seu banco, acesse o `Painel` (painel de controles) de seu provedor hospedeiro, para habilitar o **acesso remoto**. Localize a janela de banco de dados e, em seguida, clique em `MySQL Remoto` .

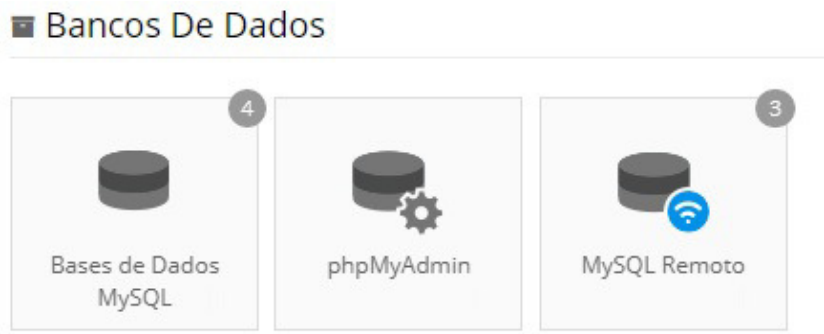

Figura 12.8: Habilitando o acesso remoto

Nessa página, o campo `Host` é quem libera a conexão ao banco de dados, e poderá ser configurado de duas maneiras:

1. Se você digitar apenas um símbolo de `%`, ele vai liberar o acesso para qualquer IP conectar em seu banco de dados MySQL.

2. Digite um número de IP específico para liberar o acesso apenas a esse usuário.

Então, clique no botão de criar. O painel de adicionar *Host* ficará desta forma:

Figura 12.9: IP remoto configurado

Para remover um IP de acesso, clique no botão de excluir conforme demonstrado na figura a seguir.

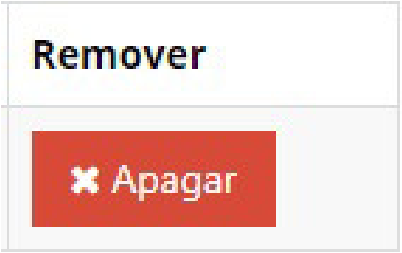

Figura 12.10: Excluindo um acesso remoto

Feita essa pequena alteração, já é possível acessar o banco de dados na internet pelo seu aplicativo. Mas não se esqueça que também é preciso existir uma pasta chamada `app_run` no seu provedor hospedeiro, com todos os arquivos em PHP adicionados dentro dela.

Quando o leitor criar o arquivo do banco de dados no seu provedor, você terá um novo nome de usuário e deverá criar uma senha para acesso. Não esqueça de alterar essas configurações nos arquivos em PHP usados. Você necessitará identificar, junto ao seu provedor, o endereço que vai usar para acessar o MySQL.

O endereço final obtido para inserir em cada componente `Web` deverá ser: http://www.seusite.com/app_run/gravar.php. Mas não se esqueça de alterar o nome de cada arquivo em PHP.

Gere o arquivo APK, conforme vimos antes, para a instalação em seu dispositivo móvel e realize os testes. Você precisará de acesso à internet para que o app funcione e transmita os dados para o servidor online.

Caso ocorra algum erro, verifique novamente os passos para deixar o arquivo com acesso remoto, principalmente se alterou a propriedade *URL* dos componentes `Web` .

12.7 RESUMINDO

No próximo capítulo, aprenderemos como disponibilizar o aplicativo na *Play Store*, para que ele se torne público e qualquer pessoa possa baixá-lo.

CAPÍTULO 13

PUBLICANDO NA PLAY STORE

Se o leitor chegou até aqui, é porque já se tornou um inventor de aplicativos Android. Agora, é hora de disponibilizar seu projeto para o mundo.

Neste capítulo final, demonstraremos como publicar seu app na loja de aplicativos da Google.

13.1 CRIANDO UMA CONTA DE DESENVOLVEDOR

Precisamos acessar o site da Google para criar uma conta de desenvolvedor de aplicativos. Para se logar, acesse o link https://play.google.com/apps/publish/. Você deverá se logar obrigatoriamente a uma conta da Google. Após informar e-mail e senha, será exibida a tela a seguir.

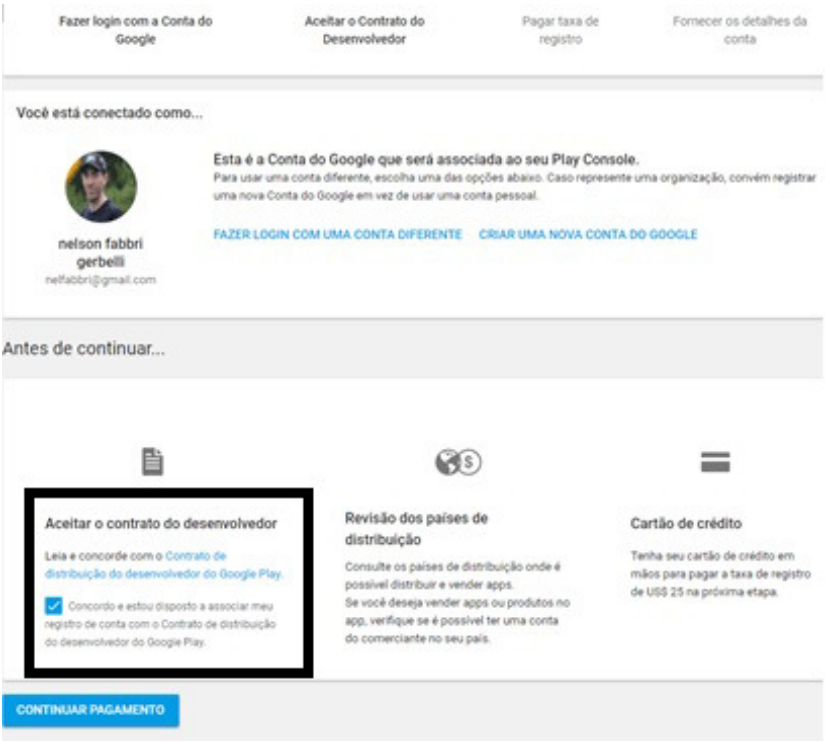
Fazer login com a Conta do Google
Aceitar o Contrato do Desenvolvedor
Pagar taxa de registro
Fornecer os detalhes da conta
Você está conectado como...
nelson fabbri gerbelli
nelfabbri@gmail.com
Esta é a Conta do Google que será associada ao seu Play Console.
Para usar uma conta diferente, escolha uma das opções abaixo. Caso represente uma organização, convém registrar uma nova Conta do Google em vez de usar uma conta pessoal.
FAZER LOGIN COM UMA CONTA DIFERENTE
CRIAR UMA NOVA CONTA DO GOOGLE
Antes de continuar...
Aceitar o contrato do desenvolvedor
Leia e concorde com o Contrato de distribuição do desenvolvedor do Google Play.
Concordo e estou disposto a associar meu registro de conta com o Contrato de distribuição do desenvolvedor do Google Play.
Revisão dos países de distribuição
Consulte os países de distribuição onde é possível distribuir e vender apps.
Se você deseja vender apps ou produtos no app, verifique se é possível ter uma conta do comerciante no seu país.
Cartão de crédito
Tenha seu cartão de crédito em mãos para pagar a taxa de registro de US$ 25 na próxima etapa.
CONTINUAR PAGAMENTO

Figura 13.1: Conta do desenvolvedor

Leia o **Contrato de distribuição do desenvolvedor do Google Play** e, se estiver de acordo, marque a opção indicada na imagem anterior. Após seu aceite, clique no botão **Continuar pagamento**.

> Existe uma taxa de registro no valor de 25 dólares. Ela precisará ser paga apenas uma vez, e o seu registro terá duração indeterminada.

Ao clicar no botão **Continuar pagamento**, será exibida uma janela para a realização do pagamento através de um cartão de crédito. Informe seus dados conforme a solicitação da tela.

Após o pagamento, você será redirecionado para completar as informações do **perfil do desenvolvedor**, conforme exibido a seguir. Preencha-o com seus dados e, ao final, clique no botão `Registro completo`.

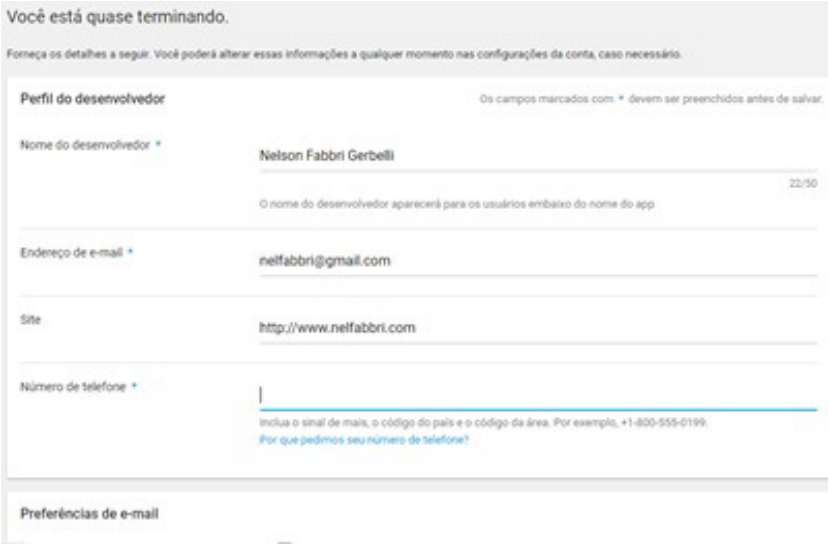
Você está quase terminando.
Forneça os detalhes a seguir. Você poderá alterar essas informações a qualquer momento nas configurações da conta, caso necessário.
Perfil do desenvolvedor
Os campos marcados com * devem ser preenchidos antes de salvar.
Nome do desenvolvedor *
Nelson Fabbri Gerbelli
22/50
O nome do desenvolvedor aparecerá para os usuários embaixo do nome do app
Endereço de e-mail *
nelfabbri@gmail.com
Site
http://www.nelfabbri.com
Número de telefone *
Inclua o sinal de mais, o código do país e o código da área. Por exemplo, +1-800-555-0199.
Por que pedimos seu número de telefone?
Preferências de e-mail

Figura 13.2: Perfil do desenvolvedor

Terminado o registro, você entrará na tela principal para começar a disponibilizar seu aplicativo. A figura a seguir exibe a tela de **boas-vindas**.

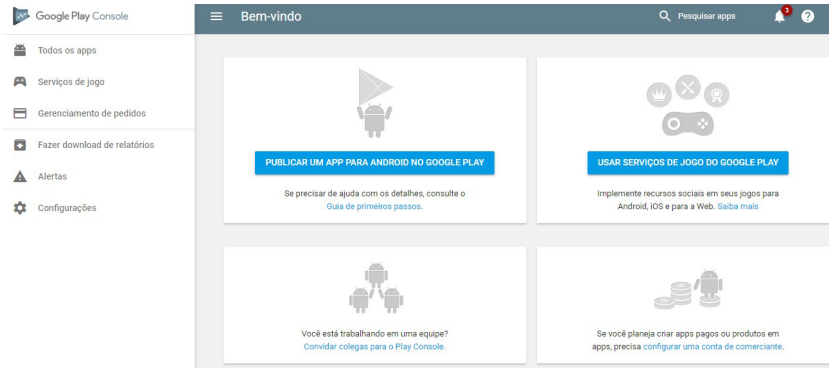
Google Play Console
Todos os apps
Serviços de jogo
Gerenciamento de pedidos
Fazer download de relatórios
Alertas
Configurações
Bem-vindo
Pesquisar apps
PUBLICAR UM APP PARA ANDROID NO GOOGLE PLAY
Se precisar de ajuda com os detalhes, consulte o Guia de primeiros passos.
USAR SERVIÇOS DE JOGO DO GOOGLE PLAY
Implemente recursos sociais em seus jogos para Android, iOS e para a Web. Saiba mais
Você está trabalhando em uma equipe? Convidar colegas para o Play Console.
Se você planeja criar apps pagos ou produtos em apps, precisa configurar uma conta de comerciante.

Figura 13.3: Tela inicial

Todo o procedimento visto anteriormente só será necessário na primeira vez que acessar a loja.

Sempre que quisermos disponibilizar um aplicativo na *Play Store*, teremos quatro etapas que precisarão ser preenchidas. São elas:

1. Versões de apps;
2. Detalhes do app;
3. Classificação de conteúdo;
4. Preços e distribuição.

13.2 INFORMANDO OS DETALHES DO APP

Para começar a publicar, primeiramente clique no botão `Publicar um app para Android no Google Play` (botão em azul) para entrar com as informações do app. Nessa página, você informará o idioma padrão e o título do seu aplicativo. Depois,

clique no botão `Criar` para prosseguir.

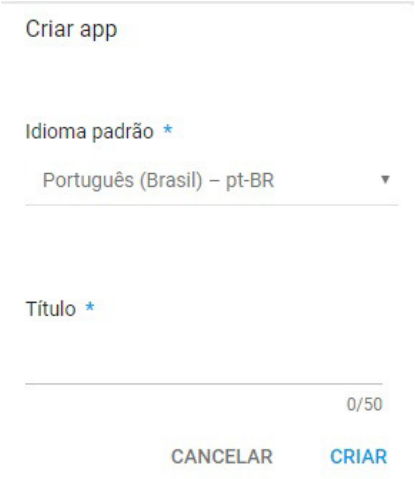

Figura 13.4: Informações do app

Na nova tela, faça uma **breve descrição** sobre as funções do aplicativo, bem como uma **descrição completa**.

Mais abaixo na página, devemos incluir as telas do seu app na seção **Detalhes do aplicativo**. Também é preciso inserir um ícone de alta qualidade e uma imagem de gráfico de recursos. Atente-se aos tamanhos necessários que estão informados na página. O ícone deverá ter exatamente 512px por 512px, e o gráfico de recursos, uma imagem que ficará como sua propaganda na loja, deverá ter 1024px de largura por 500px de altura.

Ainda na mesma página, deveremos indicar a **classificação de nosso aplicativo**, informando se ele é um jogo ou um aplicativo. Selecione a sua escolha e, logo na sequência, indique a categoria em que se encaixa.

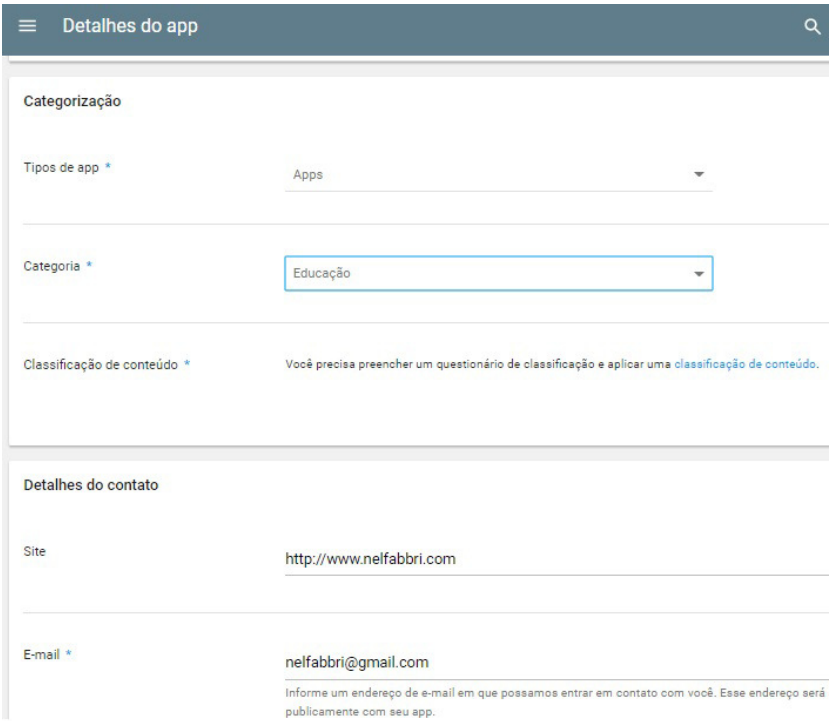
Detalhes do app
Categorização
Tipos de app *
Apps
Categoria *
Educação
Classificação de conteúdo *
Você precisa preencher um questionário de classificação e aplicar uma classificação de conteúdo.
Detalhes do contato
Site
http://www.nelfabbri.com
E-mail *
nelfabbri@gmail.com
Informe um endereço de e-mail em que possamos entrar em contato com você. Esse endereço será publicamente com seu app.

Figura 13.5: Categorização do aplicativo

No final da página, encontra-se a opção de **política de privacidade**. Faça a sua escolha para finalizar o processo dos detalhes do app. Após inserir todas as informações necessárias e obrigatórias, clique no botão `Salvar rascunho` que se encontra no topo da tela.

Verifique se o ícone da opção `Detalhes do app` , encontrado ao lado esquerdo da tela, está com um símbolo verde. Isso indica que finalizamos essa etapa e podemos prosseguir para a próxima. Se o ícone não for alterado, alguma informação obrigatória não foi cadastrada. Reveja os campos que estão marcados com um

asterisco e clique novamente no botão `Salvar rascunho`.

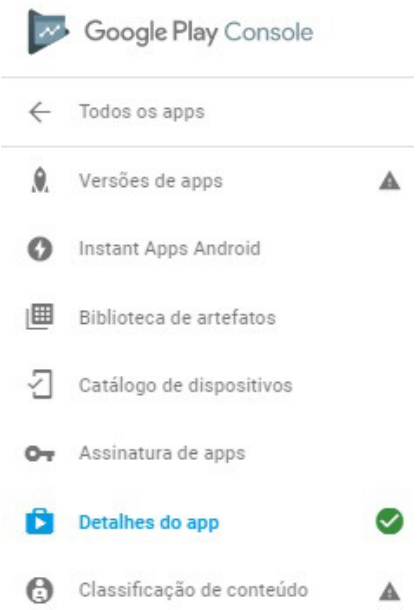

Figura 13.6: Confirmação dos detalhes do app

13.3 VERSÕES DO APP

Para enviar seu aplicativo à Play Store, precisamos acessar a opção `Versões de apps`, do lado esquerdo da tela. Veja na figura a seguir a tela que surgirá. Nela, clique no botão `Gerenciar produção` para começar o envio do arquivo APK.

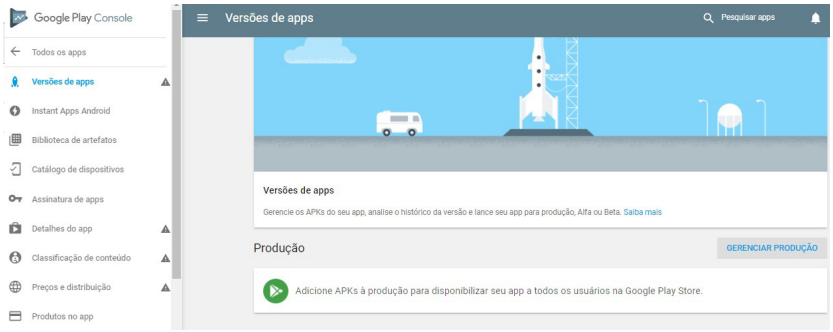
Google Play Console
Todos os apps
Versões de apps
Instant Apps Android
Biblioteca de artefatos
Catálogo de dispositivos
Assinatura de apps
Detalhes do app
Classificação de conteúdo
Preços e distribuição
Produtos no app
Versões de apps
Pesquisar apps
Versões de apps
Gerencie os APKs do seu app, analise o histórico da versão e lance seu app para produção, Alfa ou Beta. Saiba mais
Produção
GERENCIAR PRODUÇÃO
Adicione APKs à produção para disponibilizar seu app a todos os usuários na Google Play Store.

Figura 13.7: Gerenciar produção

Após o clique, uma nova tela para criar uma versão será carregada. Clique no botão de mesmo nome para continuar.

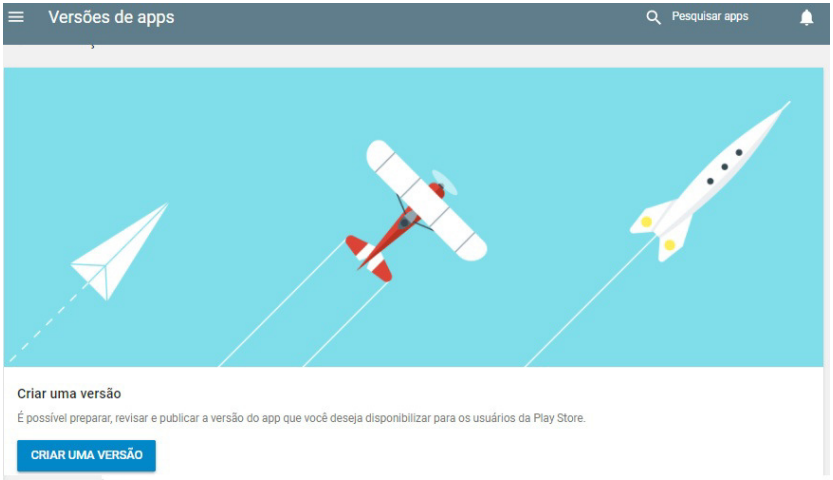
Versões de apps
Pesquisar apps
Criar uma versão
É possível preparar, revisar e publicar a versão do app que você deseja disponibilizar para os usuários da Play Store.
CRIAR UMA VERSÃO

Figura 13.8: Criar uma versão

O *Google Play App Signing* solicitará uma chave de assinatura, clique em `Recusar` para habilitar o envio de seu app.

Uma palavra sobre chave de assinatura

Durante o processo de construção do `.apk`, seu aplicativo é assinado com uma chave privada digital que está associada à sua conta. Sempre que você criar uma nova versão do seu app, essa mesma chave é usada para assinar a nova. Quando um dispositivo Android possuir seu app já instalado, ele saberá qual foi a chave usada para assiná-lo. Então, para instalar uma versão atualizada do app, o novo aplicativo deverá ser assinado pela mesma chave.

Sua chave digital privada é armazenada em um arquivo de armazenamento de chaves. Normalmente, o servidor MIT App Inventor vai criar esse arquivo quando necessário e armazená-lo para você, para que não precise se preocupar com isso. Esse é o motivo pelo qual devemos **recusar** a chave do *Google Play App Signing*.

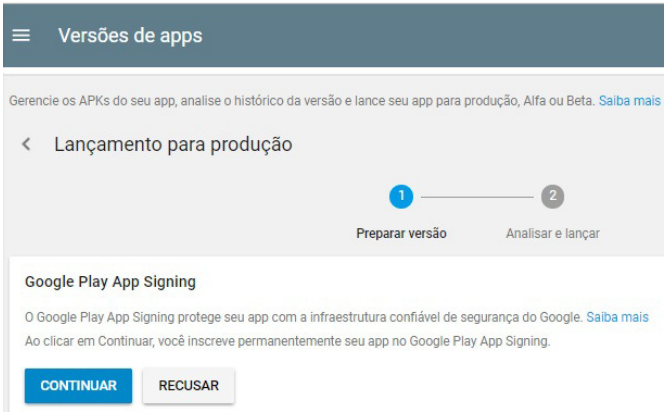
Versões de apps
Gerencie os APKs do seu app, analise o histórico da versão e lance seu app para produção, Alfa ou Beta. Saiba mais
Lançamento para produção
1
2
Preparar versão
Analisar e lançar
Google Play App Signing
O Google Play App Signing protege seu app com a infraestrutura confiável de segurança do Google. Saiba mais
Ao clicar em Continuar, você inscreve permanentemente seu app no Google Play App Signing.
CONTINUAR
RECUSAR

Figura 13.9: Google Play App Signing

Após clicar no botão de recusar, realize o upload do seu arquivo APK, clicando no botão `Enviar APK`, então aguarde o carregamento. Veja se surgiu um ícone verde confirmando a operação ao lado esquerdo da tela.

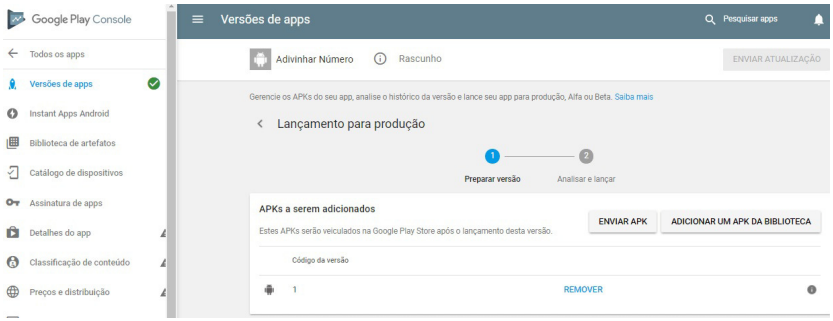

Figura 13.10: Upload completo

13.4 CLASSIFICANDO O APLICATIVO

Acesse a opção lateral `Classificação de conteúdo`. O sistema de classificação de conteúdo para apps e jogos do Google Play foi projetado para oferecer classificações reconhecidas e localmente relevantes aos usuários de todo o mundo.

Esse processo é obrigatório para que você consiga publicar seu app de acordo com as leis de cada país onde ele será disponibilizado. Esse sistema inclui classificações oficiais da Coalizão Internacional de Classificação Etária (IARC, na sigla em inglês) e de seus órgãos participantes. Após a leitura do texto, clique no botão `Continuar`.

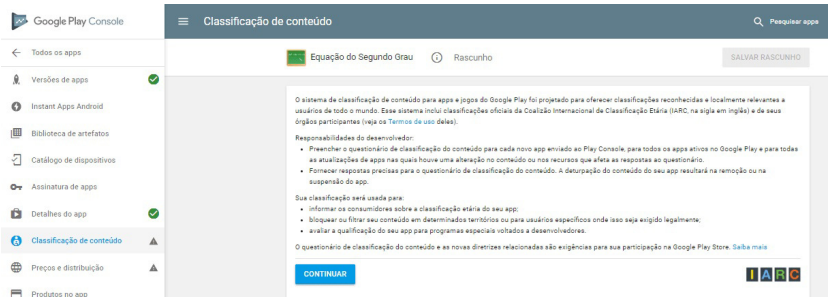
Google Play Console
Classificação de conteúdo
Pesquisar apps
Todos os apps
Versões de apps
Instant Apps Android
Biblioteca de artefatos
Catálogo de dispositivos
Assinatura de apps
Detalhes do app
Classificação de conteúdo
Preços e distribuição
Produtos no app
Equação do Segundo Grau
Rascunho
SALVAR RASCUNHO
O sistema de classificação de conteúdo para apps e jogos do Google Play foi projetado para oferecer classificações reconhecidas e localmente relevantes a usuários de todo o mundo. Esse sistema inclui classificações oficiais da Coalizão Internacional de Classificação Etária (IARC, na sigla em inglês) e de seus órgãos participantes (veja os Termos de uso deles).
Responsabilidades do desenvolvedor:
Preencher o questionário de classificação do conteúdo para cada novo app enviado ao Play Console, para todos os apps ativos no Google Play e para todas as atualizações de apps nas quais houve uma alteração no conteúdo ou nos recursos que afeta as respostas ao questionário.
Fornecer respostas precisas para o questionário de classificação do conteúdo. A deturpação do conteúdo do seu app resultará na remoção ou na suspensão do app.
Sua classificação será usada para:
informar os consumidores sobre a classificação etária do seu app;
bloquear ou filtrar seu conteúdo em determinados territórios ou para usuários específicos onde isso seja exigido legalmente;
avaliar a qualificação do seu app para programas especiais voltados a desenvolvedores.
O questionário de classificação do conteúdo e as novas diretrizes relacionadas são exigências para sua participação na Google Play Store. Saiba mais
CONTINUAR
IARC

Figura 13.11: Classificação de conteúdo

Informe seu endereço de e-mail, e depois selecione a categoria do seu app.

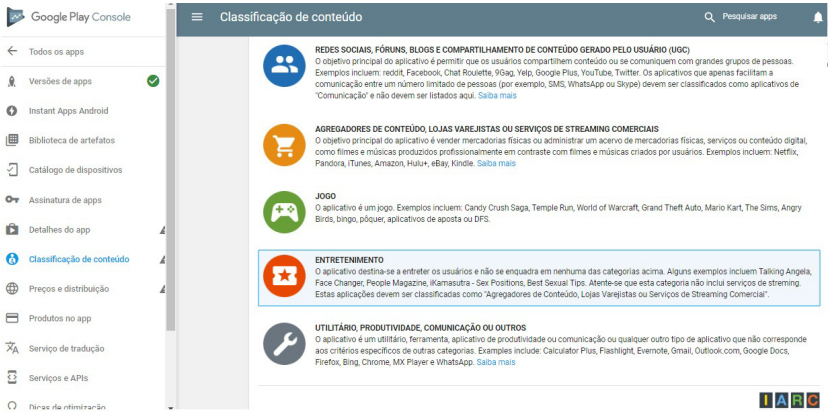
Google Play Console
Classificação de conteúdo
Pesquisar apps
Todos os apps
Versões de apps
Instant Apps Android
Biblioteca de artefatos
Catálogo de dispositivos
Assinatura de apps
Detalhes do app
Classificação de conteúdo
Preços e distribuição
Produtos no app
Serviço de tradução
Serviços e APIs
REDES SOCIAIS, FÓRUNS, BLOGS E COMPARTILHAMENTO DE CONTEÚDO GERADO PELO USUÁRIO (UGC)
O objetivo principal do aplicativo é permitir que os usuários compartilhem conteúdo ou se comuniquem com grandes grupos de pessoas. Exemplos incluem: reddit, Facebook, Chat Roulette, 9Gag, Yelp, Google Plus, YouTube, Twitter. Os aplicativos que apenas facilitam a comunicação entre um número limitado de pessoas (por exemplo, SMS, WhatsApp ou Skype) devem ser classificados como aplicativos de "Comunicação" e não devem ser listados aqui. Saiba mais
AGREGADORES DE CONTEÚDO, LOJAS VAREJISTAS OU SERVIÇOS DE STREAMING COMERCIAIS
O objetivo principal do aplicativo é vender mercadorias físicas ou administrar um acervo de mercadorias físicas, serviços ou conteúdo digital, como filmes e músicas produzidos profissionalmente em contraste com filmes e músicas criados por usuários. Exemplos incluem: Netflix, Pandora, iTunes, Amazon, Hulu+, eBay, Kindle. Saiba mais
JOGO
O aplicativo é um jogo. Exemplos incluem: Candy Crush Saga, Temple Run, World of Warcraft, Grand Theft Auto, Mario Kart, The Sims, Angry Birds, bingo, pôquer, aplicativos de aposta ou DFS.
ENTRETENIMENTO
O aplicativo destina-se a entreter os usuários e não se enquadra em nenhuma das categorias acima. Alguns exemplos incluem Talking Angela, Face Changer, People Magazine, iKamasutra - Sex Positions, Best Sexual Tips. Atente-se que esta categoria não inclui serviços de streming. Estas aplicações devem ser classificadas como "Agregadores de Conteúdo, Lojas Varejistas ou Serviços de Streaming Comercial".
UTILITÁRIO, PRODUTIVIDADE, COMUNICAÇÃO OU OUTROS
O aplicativo é um utilitário, ferramenta, aplicativo de produtividade ou comunicação ou qualquer outro tipo de aplicativo que não corresponde aos critérios específicos de outras categorias. Examples include: Calculator Plus, Flashlight, Evernote, Gmail, Outlook.com, Google Docs, Firefox, Bing, Chrome, MX Player e WhatsApp. Saiba mais
IARC

Figura 13.12: Classificação de conteúdo

Após a seleção da categoria, o leitor deverá responder ao questionário que surgirá.

Classificação de conteúdo
Pesquisar apps
O aplicativo contém inferências, referências ou representações sexuais, violência sexual, insinuação, trajes sedutores ou nudez? *
Observe que está pergunta não se refere ao conteúdo gerado pelo usuário.
Sim
Não
APOSTAS SIMULADAS, APOSTAS REAIS OU PAGAMENTOS EM DINHEIRO
O aplicativo contém apostas, simulações de jogos de cassino/bingo ou pagamentos em dinheiro reais? *
Observe que esta pergunta não se refere ao conteúdo gerado pelo usuário.
Sim
Não
LINGUAGEM
SUBSTÂNCIAS CONTROLADAS
HUMOR GROSSEIRO
DIVERSOS
CALCULAR CLASSIFICAÇÃO
SALVAR QUESTIONÁRIO
IARC

Figura 13.13: Questionário da classificação

Responda a todo o questionário, e então clique no botão `Salvar questionário`. Após seu clique, será habilitado um botão para calcular a classificação, que deverá ser clicado.

Surgirá uma página exibindo toda a classificação de seu app. No final dela, você encontra o botão `Aplicar classificação`. Clique nele.

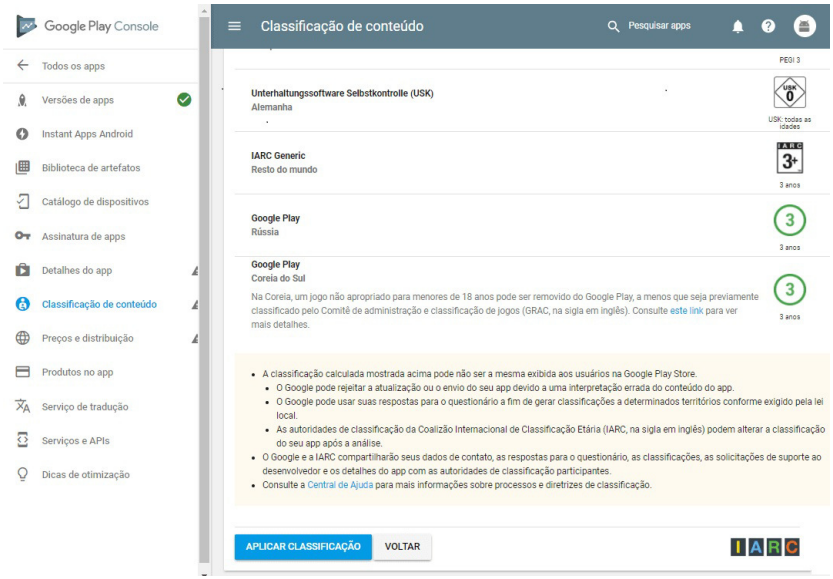
Google Play Console
Todos os apps
Versões de apps
Instant Apps Android
Biblioteca de artefatos
Catálogo de dispositivos
Assinatura de apps
Detalhes do app
Classificação de conteúdo
Preços e distribuição
Produtos no app
Serviço de tradução
Serviços e APIs
Dicas de otimização
Classificação de conteúdo
Pesquisar apps
PEGI 3
Unterhaltungssoftware Selbstkontrolle (USK)
Alemanha
USK: todas as idades
IARC Generic
Resto do mundo
3 anos
Google Play
Rússia
3 anos
Google Play
Coreia do Sul
Na Coreia, um jogo não apropriado para menores de 18 anos pode ser removido do Google Play, a menos que seja previamente classificado pelo Comitê de administração e classificação de jogos (GRAC, na sigla em inglês). Consulte este link para ver mais detalhes.
3 anos
A classificação calculada mostrada acima pode não ser a mesma exibida aos usuários na Google Play Store.
O Google pode rejeitar a atualização ou o envio do seu app devido a uma interpretação errada do conteúdo do app.
O Google pode usar suas respostas para o questionário a fim de gerar classificações a determinados territórios conforme exigido pela lei local.
As autoridades de classificação da Coalizão Internacional de Classificação Etária (IARC, na sigla em inglês) podem alterar a classificação do seu app após a análise.
O Google e a IARC compartilharão seus dados de contato, as respostas para o questionário, as classificações, as solicitações de suporte ao desenvolvedor e os detalhes do app com as autoridades de classificação participantes.
Consulte a Central de Ajuda para mais informações sobre processos e diretrizes de classificação.
APLICAR CLASSIFICAÇÃO
VOLTAR
IARC

Figura 13.14: Classificação de seu app

Verifique se o ícone `Classificação de Conteúdos` , ao lado esquerdo da tela, ficou marcado como completo, conforme demonstra a figura a seguir.

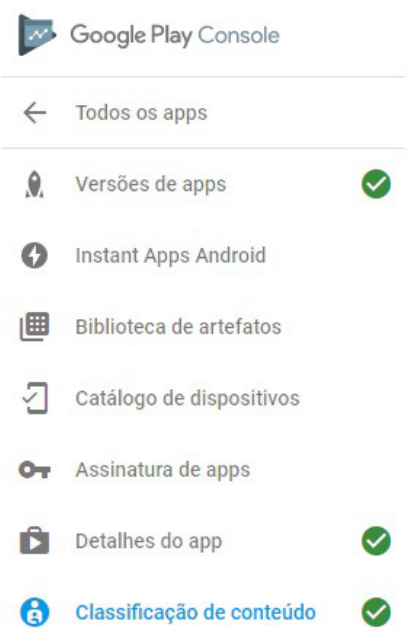

Figura 13.15: Classificação de conteúdo completa

13.5 PREÇOS E DISTRIBUIÇÃO

A última etapa que deveremos realizar é a definição dos `Preços e distribuição`. Ao selecionar essa opção no menu lateral, será exibida a tela a seguir. Informe se seu aplicativo será pago ou gratuito, e também para quais países ele deverá ser disponibilizado.

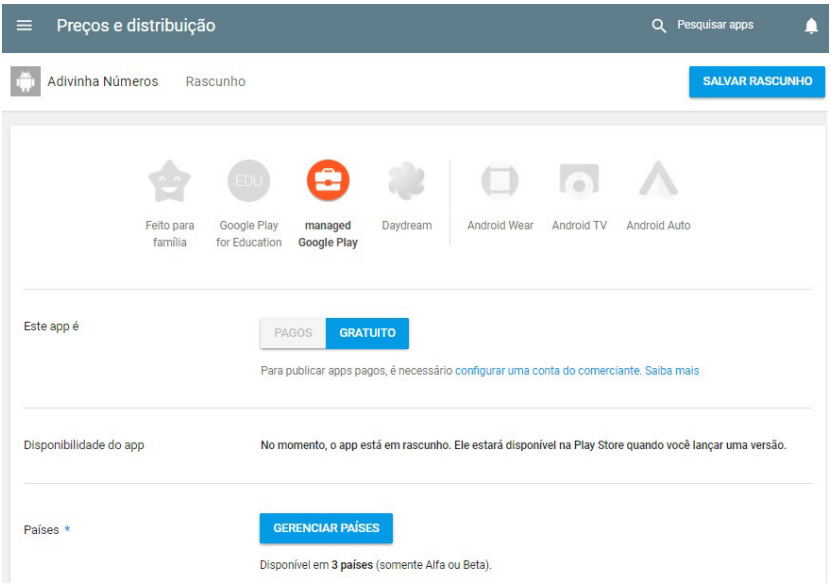

Figura 13.16: Preços e distribuição

Após a seleção dos países, indique se o seu app foi desenvolvido para o público infantil e se contém anúncios. Responda também às demais questões presentes na tela.

Ao final das respostas, vá ao topo da página e clique no botão `Salvar rascunho`. Verifique se todas as opções estão marcadas, como mostra a figura a seguir.

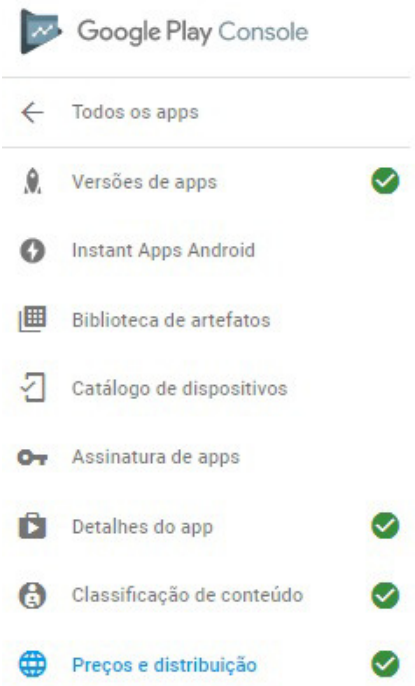

Figura 13.17: Confirmação de sucesso

Note que, ao terminar todos os passos citados, será habilitado o botão `Pronto para publicar`. Caso realmente deseje publicar, clique nesse link; caso contrário, ele ficará salvo como rascunho e poderá ter seu lançamento quando o leitor desejar. A seguinte tela será exibida caso seja clicado no botão `Pronto para publicar`.

Seu app está pronto para ser publicado
Agora você pode publicar seu app começando o lançamento de uma versão na página "Gerenciar versões".
DISPENSAR
GERENCIAR VERSÕES

Figura 13.18: App pronto para publicação

Clique em `Gerenciar versões` para ir à página de lançamento. Verifique uma informação dizendo que *você tem uma versão em produção que não foi lançada* e clique no botão `Editar versão` para continuar com o lançamento:

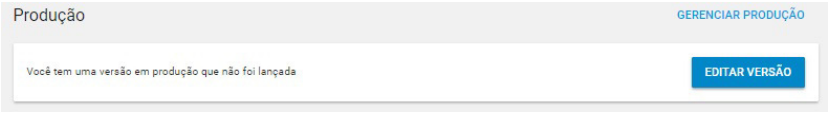

Figura 13.19: Produção

Role até o final da nova página que surgiu, e clique no botão `Revisar` para habilitar a opção de `Iniciar lançamento para produção`, que deverá ser clicada.

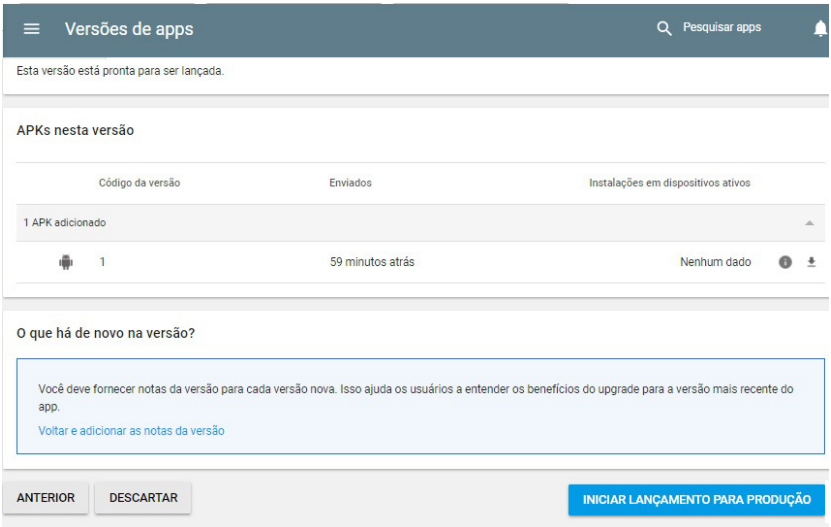
Versões de apps
Pesquisar apps
Esta versão está pronta para ser lançada.
APKs nesta versão
Código da versão
Enviados
Instalações em dispositivos ativos
1 APK adicionado
1
59 minutos atrás
Nenhum dado
O que há de novo na versão?
Você deve fornecer notas da versão para cada versão nova. Isso ajuda os usuários a entender os benefícios do upgrade para a versão mais recente do app.
Voltar e adicionar as notas da versão
ANTERIOR
DESCARTAR
INICIAR LANÇAMENTO PARA PRODUÇÃO

Figura 13.20: Iniciar lançamento

Então, a seguinte imagem será exibida, informando que seu app será disponibilizado para todos os usuários da Play Store. Clique em `Confirmar` para prosseguir.

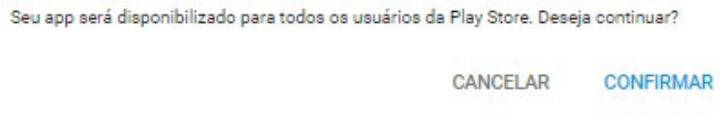

Figura 13.21: Iniciar lançamento

Após a confirmação, será dada sequência à publicação, indicando que seu app está com a **Publicação Pendente**:

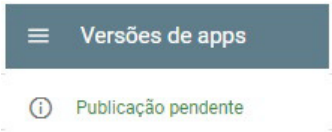

Figura 13.22: Publicação pendente

Pronto, agora é só aguardar alguns instantes que seu app logo estará disponível para download.

Capítulo 14

CONCLUSÃO

Tivemos o prazer de apresentar ao leitor uma ferramenta para o desenvolvimento de aplicativos para o sistema operacional Android. Utilizamos o ambiente do MIT App Inventor, local onde arrastamos e soltamos blocos para criar nossos aplicativos sugeridos.

Em uma primeira parte do livro, logo nos capítulos iniciais, realizamos um verdadeiro tour pelo App Inventor, conhecendo as seções do ambiente de desenvolvimento, paletes de componentes, área de viewer, seção de mídia, propriedades e a área dos blocos, onde sempre encontramos os blocos para realizar os passos necessários para solucionar um problema apresentado. Começamos desenvolvendo um aplicativo básico para exibir uma simples mensagem na tela do dispositivo e aprendemos as diversas formas de testá-lo, emulando no computador, celular ou tablet, ou mesmo instalando em um dispositivo.

Vimos como trabalhar com o layout dos apps e como criar blocos de variáveis e de operações matemáticas com o aplicativo **Uma simples calculadora**. Já no app **Etanol ou Gasolina?**, aprendemos a usar os blocos de decisões após a realização de cálculos e também como podemos acrescentar e acessar uma segunda tela em um aplicativo.

Aprendemos a utilizar uma API através do desenvolvimento do app **Tradutor online**, reconhecendo a voz do usuário, fazendo com que o dispositivo realizasse a leitura da frase traduzida em um dos possíveis idiomas selecionados e também compartilhasse a tradução com contatos do WhatsApp.

Vimos como usar o geolocalizador do dispositivo para identificar o seu local e, pelo app desenvolvido **Onde estacionei?**, pudemos nos conectar a uma API da Google para traçar a rota entre dois pontos distintos através da latitude e longitude identificadas.

Já em uma segunda parte do livro, começamos a trabalhar com aplicativos acessando banco de dados. Primeiramente, vimos como instalar e deixar em funcionamento um servidor web local, através do aplicativo USBWebserver e, na sequência, criamos o banco de dados do projeto de **Controle de corridas** pelo phpMyAdmin.

Com o servidor e o banco de dados prontos, vimos como incluir, consultar, excluir e alterar registros nesse banco, armazenando-os localmente. Posteriormente, aprendemos como deixar seu app com acesso remoto a um servidor hospedeiro na web, tarefa fundamental para podermos compartilhar e distribuir nosso aplicativo. E por falar em distribuir um app, no capítulo anterior, vimos como criar uma conta de desenvolvedor e como disponibilizar o aplicativo na loja da Google, a Play Store.

Neste livro, apresentamos uma excelente opção de desenvolvimento de aplicativos. O leitor poderá desenvolver apps para estudos em qualquer nível de educação, bem como para desenvolvimento de aplicativos pessoais ou mesmo profissionais.

Deixo aqui alguns sites e fóruns interessantes sobre o App Inventor, para que o leitor possa se aprimorar na arte de desenvolvimento de aplicativos e conhecer novos recursos. Infelizmente, não existe muita literatura em português.

- http://appinventorbrasil.com.br/ — Site em português com um bom material sobre o App Inventor.
- https://www.androidpt.com/forum/61-app-inventor/ — Fórum específico da Comunidade Portuguesa de Android.
- http://appinventor.mit.edu/explore/app-month-gallery.html — Galeria de aplicativos premiados do MIT App Inventor, em inglês.
- https://groups.google.com/forum/#!forum/mitappinventortest — Grupo do Google sobre desenvolvimento, em inglês.
- https://groups.google.com/forum/#!categories/mitappinventortest/mit-appinventor-2 — Outro grupo do Google sobre desenvolvimento, em inglês.
- http://kio4.com/appinventor/index.htm — Site com muitos exemplos para baixar e estudar, em espanhol.

A partir de agora, você está no comando e já pode ser considerado um **inventor de aplicativos** Android. Basta colocar a sua imaginação para funcionar.

Bons trabalhos!